AD-A054 310 COMMAND AND CONTROL TECHNICAL CENTER WASHINGTON D C F/G 15/7
THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK) PROGR--ETC(U)
JUN 77 D J SANDERS, P F MAYKRANTZ, J M HERRIN
UNCLASSIFIED CCTC-CSM-MM-9-77-V1-PT-2 SBIE-AD-E100 052 NL

1 OF 6
AD
A054310

FOR FURTHER TRAN AD-E100 052

COMPUTER SYSTEM MANUAL
CSM MM 9-77
VOLUME I, PART II
1 JUNE 1977

Vol 1 Pt-1
A054 377

AD A054310

C
C
T
C

COMMAND & CONTROL TECHNICAL CENTER

D D C
RECEIVED
MAY 25 1978
B

AD No.
DDC FILE COPY

Vol-2
A050081

THE CCTC QUICK-REACTING WAR GAMING SYSTEM (QUICK)

PROGRAM MAINTENANCE MANUAL

VOLUME I - DATA MANAGEMENT SUBSYSTEM

DEFENSE
COMMUNICATIONS
AGENCY

RECORD OF CHANGES

CHANGE NUMBER	DATED	DATE ENTERED	SIGNATURE OF PERSON MAKING CHANGE

CCTC should be corporate author per telecon w/ CCTC Proj. Officer C. G. Thompson, 6/16/78

JMC/RD

MB Bonnett, DCA was notified per Off instructions

(14) CCTC-CSM-MM-9-77-V1-PT-2

COMMAND AND CONTROL TECHNICAL CENTER

(18) SBIE (19) AD-E100 052

(9) Computer System Manual CSM-MM-9-77

(11) 1 June 1977

(6) THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM

(QUICK)

(12) 479p.

Program Maintenance Manual.

Volume I, Data Management Subsystem

Part II.

SUBMITTED BY:

C. G. Thompson

C. G. THOMPSON
Project Officer

APPROVED BY:

R E Harshbarger

R. E. HARSHBARGER
Acting Deputy Director
NMCS ADP

(10) Dale J./Sanders, Paul F. M./Maykrantz,
Jim M. /Herrin Edward F./Bersson

Copies of this document may be obtained from the Defense Documentation Center, Cameron Station, Alexandria, Virginia 22314.

ACKNOWLEDGMENT

This documentation was prepared under the direction of the Chief for Military Studies and Analysis, CCTC, in response to a requirement of the Studies, Analysis, and Gaming Agency, Organization of the Joint Chiefs of Staff. Technical support was provided by System Sciences, Incorporated under Contract Number DCA100-75-C-0019.

CONTENTS

Part I

Part II

Section		Page
	7.3 Output	573
	7.4 Concept of Operation	573
	7.5 Identification of Subroutine Functions	573
	7.6 Common Blocks	573
	7.7 Subroutine ENTMOD	575
8.	EXTERNAL INTERFACE MODULE (EIM)	581
	8.1 Purpose	581
	8.2 Input	581
	8.3 Output	581
	8.4 Concept of Operation	581
	8.5 Identification of Subroutine Functions	581
	8.5.1 Subroutine SIDAC	581
	8.5.2 Subroutine TABLE	589
	8.5.3 Subroutine BLDOTH	589
	8.5.4 Subroutine PLOTDATA	589
	8.6 Common Blocks	589
	8.7 Subroutine ENTMOD	594
	8.7.1 Subroutine CONVLL	597
	8.8 Subroutine BLDOTH	599
	8.8.1 Subroutine XEDEFN	628
	8.9 Subroutine PLOTDATA	633
	8.9.1 Subroutine PICS	651
	8.9.2 Subroutine PROJCT (Entry PROJCT) (Entry PROJT2)	655
	8.10 Subroutine SIDAC	658
	8.11 Subroutine TABBLE	662
9.	GENERAL UTILITIES	677
	9.1 Purpose	677
	9.2 Subroutine ABORT	683
	9.3 Subroutine ATFNDR	685
	9.4 Function ATN2PI	693
	9.5 Subroutine CINSGET	695
	9.6 Function DELLONG	697
	9.7 Function DIFFLONG	699
	9.8 Function DISTF	701
	9.9 Subroutine DOTLINE	703
	9.10 Subroutine FINDCLAS	705
	9.11 Subroutine FINDSIDE	707
	9.12 Subroutine FORMAK	709
	9.13 Function GETCLOCK	711
	9.14 Function GETDATE	713
	9.15 Subroutine GETNXT	715
	9.16 Subroutine GETSTR	719

ILLUSTRATIONS (PART II)

Figure		Page

6

Figure		Page
174	Subroutine REORDER..................................	812
175	Function RLBRT......................................	815
176	Subroutine SETORD	818
177	Subroutine SETSCH...................................	822
178	Function SLOG.......................................	831
179	Subroutine SORTIT...................................	833
180	Function SSKPC......................................	838
181	Function STD..	841
182	Subroutine SVTP.....................................	843
183	Subroutine TICMAKE..................................	849
184	Function TIMEDAY....................................	852
185	Subroutine TIMEME...................................	855
186	Subroutine TOFM.....................................	857
187	Subroutine UNCODE...................................	859
188	Subroutine VALFND...................................	862
189	Function VALTAR.....................................	871
190	Function XCOORD.....................................	873
191	Subroutine XMATH....................................	875
192	Subroutine XWHERE...................................	879
193	Function ZTAN.......................................	889
194	H* Creation From Source.............................	896
195	H* Creation From Object.............................	902
196	Utility Library Creation............................	905
197	Program PERFORM.....................................	912
198	Subroutine READIN...................................	921

TABLES (PART II)

ABSTRACT

The computerized Quick-Reacting General War Gaming System (QUICK) will accept input data, automatically generate global strategic nuclear war plans, provide output summaries, and produce tapes to simulator subsystems external to QUICK. QUICK has been programmed in FORTRAN for use on the CCTC HIS 6000 computer system.

The QUICK Maintenance Manual consists of four volumes: Volume I, Data Management Subsystem; Volume II, Weapon/Target Identification Subsystem; Volume III, Weapon Allocation Subsystem, Volume IV, Sortie Generation Subsystem. The Maintenance Manual complements the other QUICK Computer System Manuals to facilitate application of the war gaming system. This volume, Volume I in two parts, provides the programmer/analyst with a technical description of the purpose, functions, general procedures, and programming techniques applicable to the modules (programs) and subroutines of the Data Management subsystem. Companion documents are:

a. USERS MANUAL
   Computer System Manual CSM UM 9-77, Volume I
   Computer System Manual CSM UM 9-77, Volume II
   Computer System Manual CSM UM 9-74, Volume III
   Computer System Manual CSM UM 9-74, Volume IV
   Provides detailed instructions for applications of the system

b. TECHNICAL MEMORANDUM
   Technical Memorandum TM 153-77
   Provides a nontechnical description of the system for senior management personnel

SECTION 6. REPORT MODULE

6.1 Purpose

The purpose of the REPORT module is to give the user/analyst the capability to produce ad hoc print reports. REPORT is capable of addressing any set of attributes in the QUICK integrated data base and displaying them, plus any arithemetic calculations upon them, in virtually any format the user desires.

6.2 Input

The precondition of the data base required is the obvious one that non-entered data cannot be displayed. The PRINT verb does require the prior execution of a DESIGN verb so that a display table exists in the data base.

6.3 Output

Execution of a DESIGN or an ALTER verb will cause a display table to be built and retained within the data base. The composition of a display table is shown in table 18.

6.4 Concept of Operation

REPORT recognizes three verbs: DESIGN, PRINT, and ALTER.

The DESIGN verb is used to specify the format and sort to be used to display the data, the calculated variables to be included and the subset of the data base to be used. Further the user may give the display a name so that it is retained in the data base.

The PRINT verb produces the display the user has designed. In addition, the user may specify a different data base with the PRINT verb.

The ALTER verb is used to make changes to an existing saved 'display' or to construct a new 'display' from a previous one.

Subroutines (or overlays) DESIGN or ALTER build a set of utility tables which may be used to create a display. Upon table construction, subroutine DSPMAK is called to create the display. For the PRINT verb input, subroutine PRINCE will control the printing of the display desired.

As with other modules within the Data Management subsystem, data input varies according to the user's needs and is generalized to the degree possible. Since inputs do permit variability, an understanding of the implemented code can only be possible if the techniques of input definitions are understood. Section 7 of the Users Manual, Volume I, details

Table 18. Display Table Composition (Part 1 of 2)

A display table consists of the following elements the exact length and composition of most elements varies from table-to-table.

ELEMENT	DESCRIPTION
1	The display header - contents of common block DSPHED (see table 19)
2	DEFINE Names. Two words for each DEFINE
3	DEFINE Type. One word for each DEFINE =1, Normal =2, Special =3, Sum =4, Product
4	DEFINE Mode. One word for each DEFINE =1, Integer =2, Floating point
5	DEFINE Print/Sort record position
6	Number of words in execution instructions for DEFINE (one word for each DEFINE)
7	Execution instructions for DEFINEs. One set of instructions for each DEFINE the length of which is defined in element 6
8	Attribute common block address. One word for each display attribute
9	Attribute Print/Sort record position. One word for each display attribute
10	Sort Scheme. Length of the scheme is shown in element 1. A sort scheme is a sequence of sets of three words. Word 1 = 1, ascending = 2, descending Word 2 = Print/Sort record position Word 3 = 1, Integer 2, Alphabetic 3, Floating Point The order of the triples is major to minor

Table 18. (Part 2 of 2)

ELEMENT	DESCRIPTION
11	WHERE Clause. Length of the clause is shown in element 1
12	Retrieval Scheme. Length of the scheme is shown in element 1
13	Print Scheme. Length of the scheme is shown in element 1. A print scheme is a sequence of items which are of two types: page and nonpage. A page item has four words: Word 1 = 1 Word 2 = Number of headers on page Word 3 = Number of trailers on page Word 4 = Number of physical lines produced by trailers A nonpage item has a variable number of words: Word 1 = 2, for header = 3, for trailer = 4, for line which produces a single logical line = 5, for line which produces a number of logical lines Word 2 = Number of print elements Word 3 = Number of words in the format Word 4 = Number of physical lines per logical line Word 4 is followed by two sets of words. Each set has one entry for each print element First set = 1, attribute = 2, DEFINE = 3, Page Number Second set - Print/sort record position for attribute - Identifying number for DEFINE - Zero for page number These sets are followed by the format
14	Input instructions for DEFINE clauses. Total length is shown in element 1
15	SORT clause. Length is shown in element 1
16	FORMAT clause. Length is shown in element 1

all inputs. Special comments concerning the construction of the implemented print scheme and the use of 'utility tables' follows.

6.4.1 Print Scheme. The print scheme is created by DSPMAK and executed by PRINCE. Its form is outlined in table 18 (element 13). DSPMAK builds the scheme from the FORMAT clause. This clause has a series of identifying 'items' labeled PAGE, HEADER, TRAILER, and LINE. Of these, PAGE defines a logical set of the others up to the next PAGE item. If no PAGE item exists, all HEADER, TRAILER and LINE items are said to be part of one PAGE. Each HEADER, TRAILER or LINE is followed by a series of phrases which are one of the following:

- attribute
- DEFINE name
- IN (followed by a FORTRAN format, i.e., F10.2)
- alphabetic
- long string
- numeric (followed by the special words X or SPACES)
- PAGENO, a special word

These phrases are used to create the print element tables (element 13 of table 18) and the item's format. Print element tables are formed from attributes, DEFINE names and PAGENO. The tables identify the element and give the information necessary to retrieve the data value that is to be displayed. An attribute causes an addition to the format in one of three ways. First, if it is followed by an 'IN' phrase, the phrase provides the format to be inserted. Second, if the upper and lower edit ranges differ, the upper is used as a measure of the attribute's size and a format added accordingly. Third, one of the default formats (A7, I7, or F9.3) is used. A DEFINE name is treated similarly except that the second option is not available. Alphabetics and long strings both are treated in a similar fashion. The length of the string is used to create an 'H' type format (i.e., 30X). If either an alphabetic, long string or space format exceeds the line length, it is truncated. Finally a PAGENO special word adds the format I3.

In the process of building the print element tables and format for a LINE item it is determined whether the LINE is a multiple or single line. If it contains an attribute or a normal DEFINE it is a multiple. DEFINEs are of the following types:

- Sums - contain reference to their own name in an add or subtract operation

- Products - contain reference to their own name in a multiply, divide or power operation

- Specials - contain no reference to their own name but reference a Sum or Product

- Normal - all other DEFINEs

6.4.2 Utility Tables. The subroutine TABMNT creates, maintains, and deletes a set of utility tables. These are arrays which TABMNT can keep internally for a maximum of 100 words. If more space is required for the table, the current 100 words are moved into common block C40 and a TABLEZ record created. When data is desired from a particular table, an index is used to determine which set of 100 words contains the desired data. The appropriate TABLEZ record is retrieved if necessary and the data moved from C40 to internal storage. TABMNT can maintain up to five utility tables each of which can contain a maximum of 1,000 words. The tables are identified by a number from one to five (i.e., 'utility table 1'). Both DESIGN and ALTER use these tables to pass clauses to DSPMAK. PRINT uses the tables to build a new WHERE clause and to retain headers and trailers while executing the print scheme.

6.5 Identification of Subroutine Functions

6.5.1 Subroutine DESIGN. This subroutine (or overlay) carries out the function of building a new set of display specifications from scratch. First the DISPLAY clause is read and the display table set for construction. The SETTING clause is used to set values. Each DEFINE is scanned for errors, and is stored in a table. The WHERE, SORT, and FORMAT clauses are also scanned for errors and saved (see figure 79).

6.5.2 Subroutine ALTER. This subroutine (or overlay) makes alterations to old displays or constructs a new display based upon an old one. First the DISPLAY clause is used to find the old display. Then each clause of the old display is either saved in the utility tables or replaced by a new clause which has been input. The old FORMAT clause may have portions modified according to user instructions. Finally, the new display table is set up for creation or the old one deleted and reset depending upon the DISPLAY clause option selected (see figure 80).

6.5.3 Subroutine DSPMAK. This subroutine takes the utility tables built by either the DESIGN or ALTER subroutines and uses them to construct a display. First all clauses are scanned for attributes. Then the attributes collected are used to build a retrieval scheme. (For details of this process see section 4.4.) DEFINEs are placed in proper execution order and their execution tables (table 18, elements 2-7) are built. The WHERE clause is scanned for DEFINE variables and altered accordingly. The sort scheme and print scheme are now built. Finally, all the constructed elements are stored in the data base as a display table (see figure 81).

6.5.4 Subroutine PRINCE. This subroutine prints a specified display. First the DISPLAY clause is used to find the desired display table and the various schemes and tables are read from the display table. If a new WHERE clause has been input it replaces any old one. The retrieval

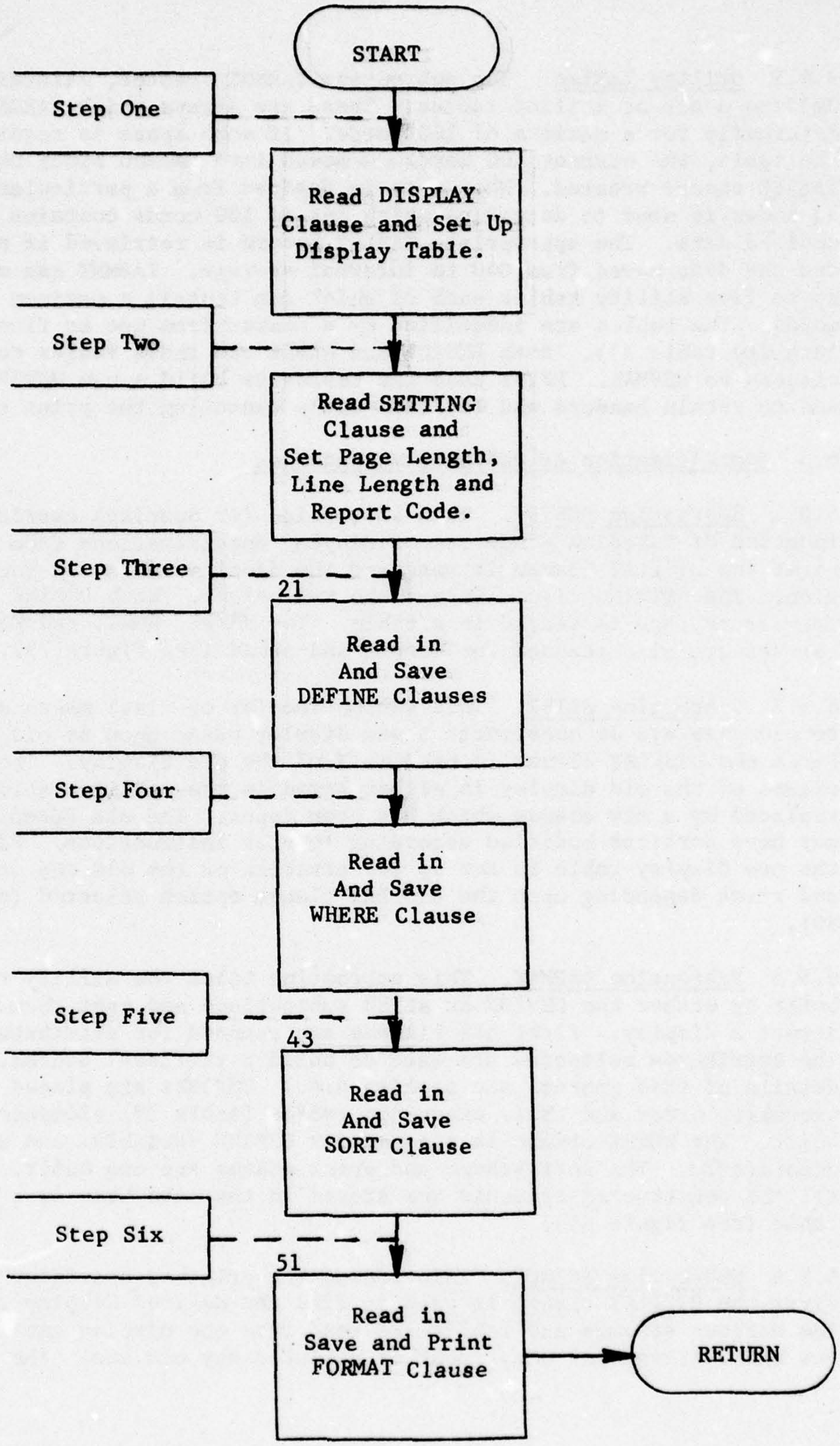


Figure 79. Subroutine DESIGN Macro Flow

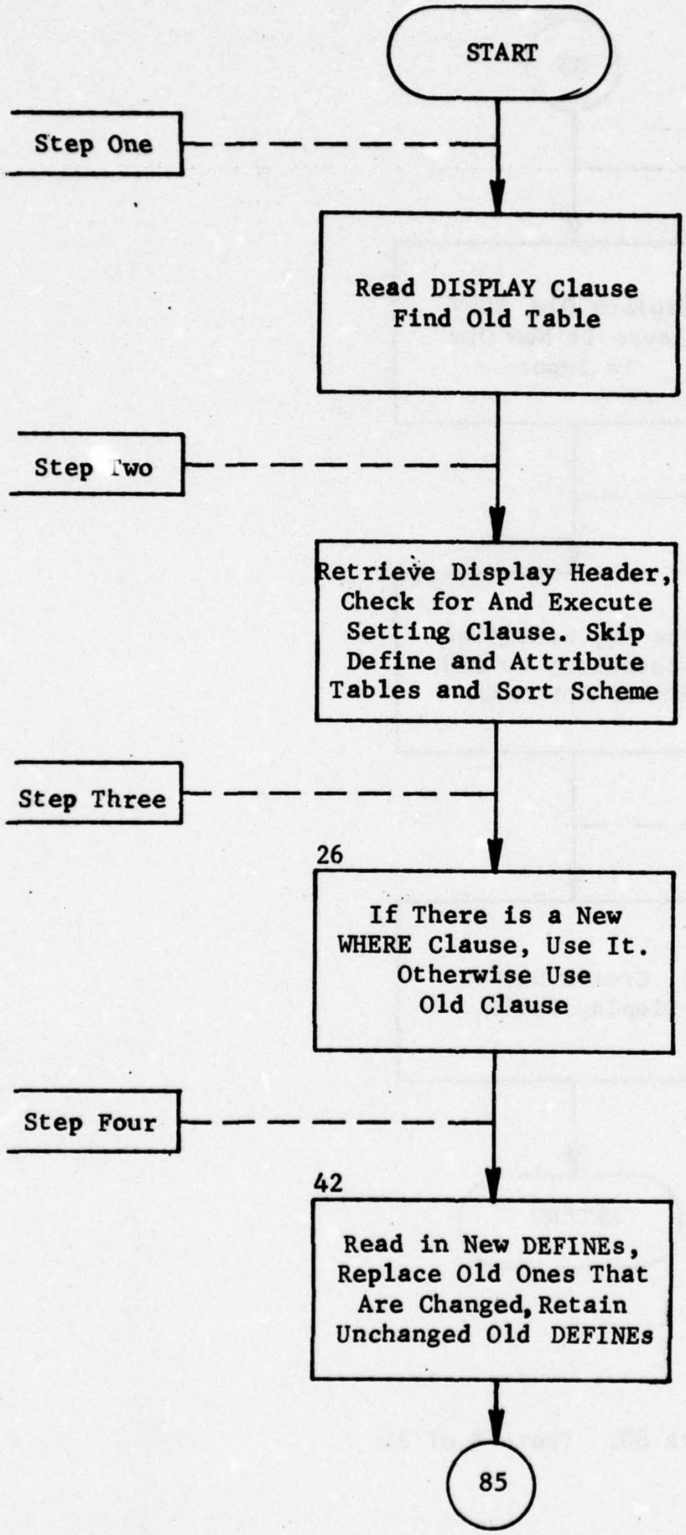


Figure 80. Subroutine ALTER Macro Flow (Part 1 of 2)

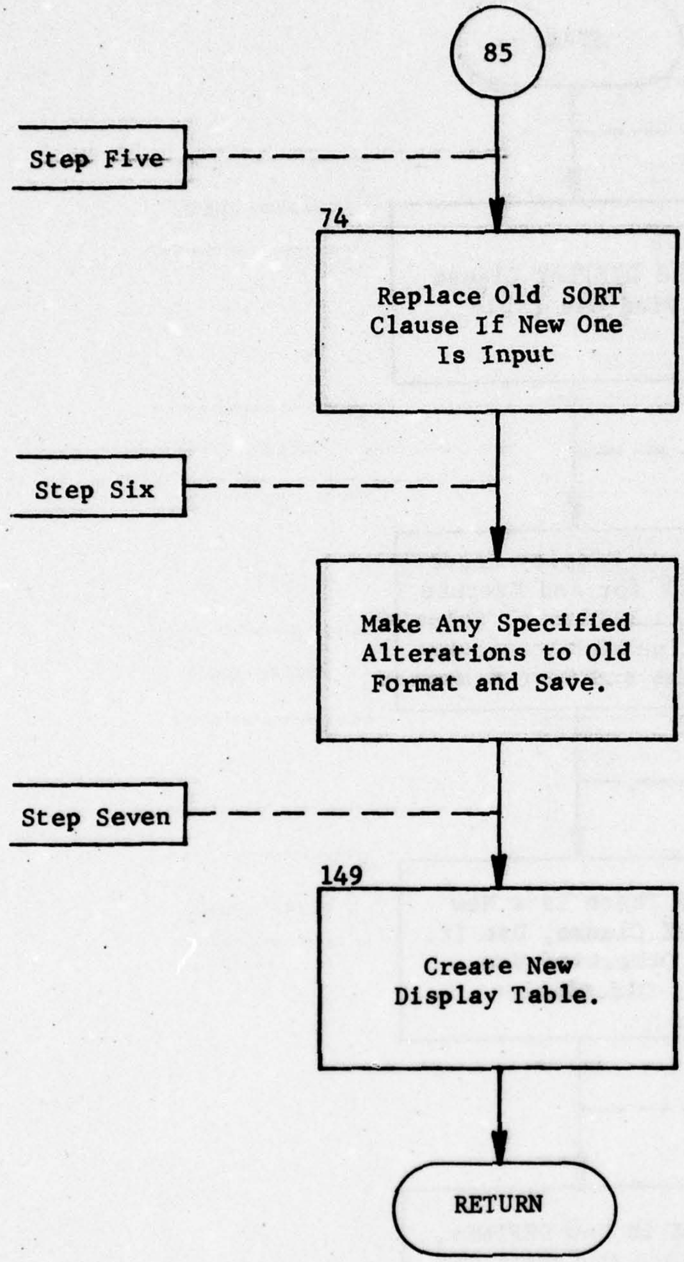


Figure 80. (Part 2 of 2)

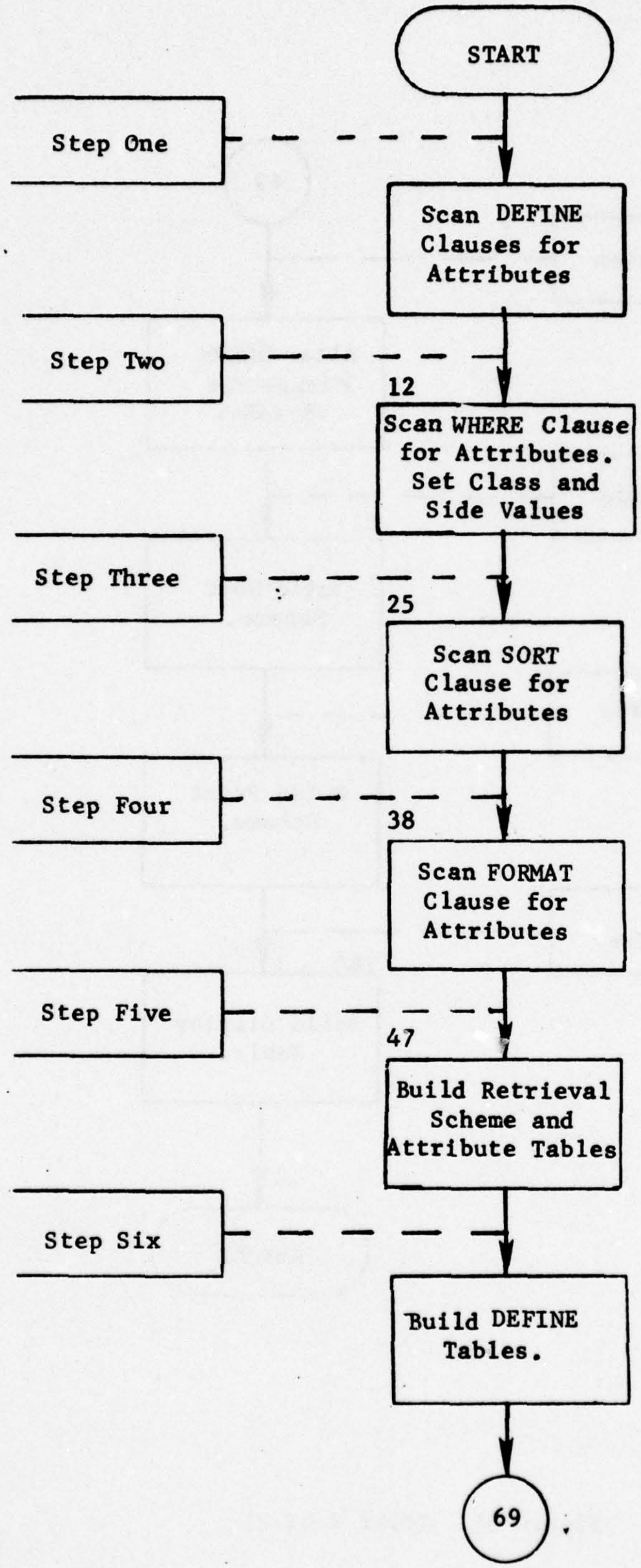


Figure 81. Subroutine DSPMAK Macro Flow (Part 1 of 2)

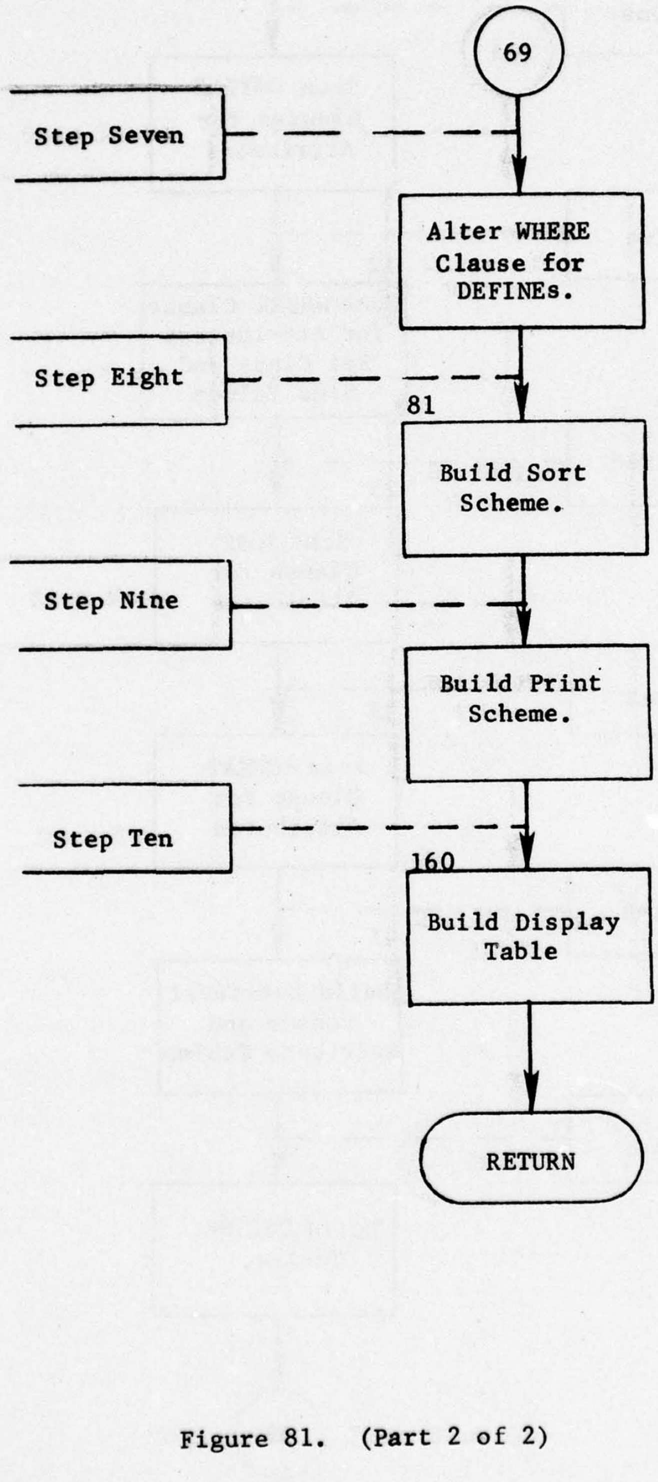


Figure 81. (Part 2 of 2)

scheme is used to build a file of the desired information. This file is then sorted if a sort has been requested. Finally, the print scheme is executed to produce the desired report (see figure 82).

6.6 Common Blocks

Common blocks which are used internally by the REPORT module are displayed in table 19.

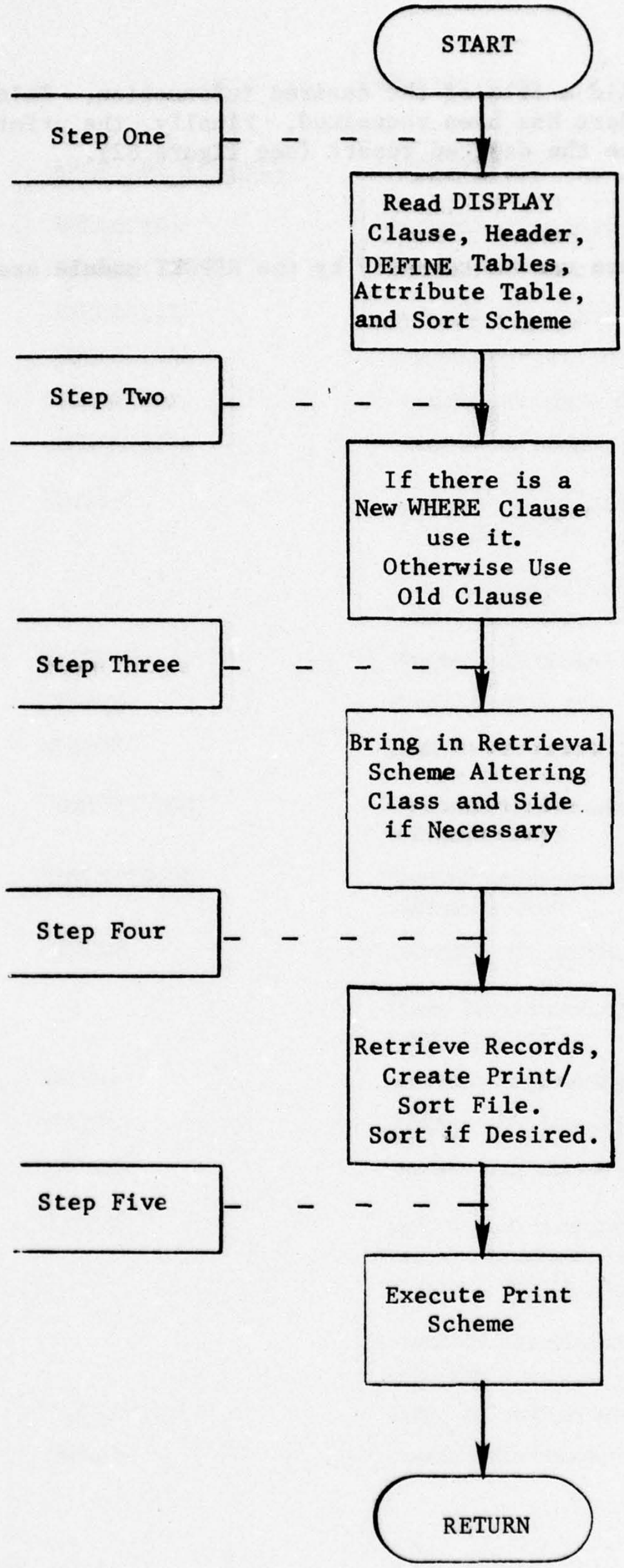


Figure 82. Subroutine PRINCE Macro Flow

Table 19. REPORT Module Internal Common Blocks (Part 1 of 3)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
ATLST		Provides communication with the ATFNDR utility subroutine
	ATNUMB(100)	Attribute's identifying number
	ATADD(100)	Attribute's address
	ATTYP(100)	Attribute's mode (=1, integer, =2, alphabetic, =3, floating point)
	ATRA(100)	Attribute's lower limit
	ATRB(100)	Attribute's upper limit
	NUMAT	Number of attributes
DEFNMZ	DFNAME(100)	DEFINE variable name
	DFPNT(100)	Pointer to DEFINE clause in utility table 1
DEFVAR	VARXX(100)	Value of DEFINE variable
DSPFRM		Provides communication with FORMAK utility subroutine
	FORMAR(50)	*Format being constructed*
	IFPNT	Pointer to next character
	IFLNG	Number of words used
DSPHED	IDHEAD(15)	Display header, each word is a separate value as follows:
	IDHEAD(1)	Page length (lines)
	IDHEAD(2)	Line length (characters)
	IDHEAD(3)	Report code
	IDHEAD(4)	Number of DEFINE variables
	IDHEAD(5)	Number of attributes
	IDHEAD(6)	Length of sort scheme (words)
	IDHEAD(7)	Length of WHERE clause (words)
	IDHEAD(8)	Primary header record type number
	IDHEAD(9)	Length of retrieval scheme (words)

Table 19. (Part 2 of 3)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
DSPHED (cont.)	IDHEAD(10)	Length of print/sort record (words)
	IDHEAD(11)	Number of pages
	IDHEAD(12)	Length of print scheme (words)
	IDHEAD(13)	Length of all DEFINE clauses (words)
	IDHEAD(14)	Length of SORT clause (words)
	IDHEAD(15)	Length of FORMAT clause (words)
IDPT	IDISPT	Pointer to next variable word in display table
NONPAG		Used internally by PRINCE for core position only
	NITEMS	Number of items in line
	NFORMW	Number of words in format
	NLINES	Number of print lines produced
ORDER	SCHORD(100)	Record type numbers in retrieval scheme order
	SORDNM(100)	Record type names in retrieval scheme order
	LENSCH	Length, in words, of retrieval scheme
PAGPAG		Used internally by PRINCE for core position only
	NUMHD	Number of headers in page
	NUMTRL	Number of trailers in page
	NTRLLN	Number of lines produced by trailers
PRINSP	PRINON	Switch to control optional prints =True, produce print =False, do not produce print
PSCOM		Used to communicate with PSREC utility
	RECORD(100)	Body of print/sort record
	RECLEN	Number of words in record

Table 19. (Part 3 of 3)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
RTLST		Used to communicate with ATFNDR utility subroutine
	RTLIST(100)	List of record type numbers for retrieval scheme
	NUMREC	Number of record types in RTLIST
	HDREC	Record type name of primary header
	HCLASS	CLASS value of primary header
	HSIDE	SIDE value of primary header
	HOPT	CLASS/SIDE option for scheme
	JHDR	Record type number of primary header
SCHEME	POINT	Points to current instruction of retrieval scheme
	SCHEME(200)	Retrieval scheme (see UMI, section 5.3.2.4)
SORSCH	SRTSCH(100)	Sort scheme (see section 6.10)
ZEES		Used internally by several routines for core position only
	ZA ZB ZC ZD ZE	Equivalent to output array from INSGET

6.7 Subroutine ENTMOD

PURPOSE:	Entry module for REPORT
ENTRY POINTS:	ENTMOD (first subroutine called when overlay REPORT is executed)
FORMAL PARAMETERS:	None
COMMON BLOCKS:	DEFNMZ, DSPHED, OOPS, PRINSP
SUBROUTINES CALLED:	ALTER, DESIGN, DSPMAK, INSGET, PRINCE
CALLED BY:	MODGET

Method:

The first step is to obtain the verb from INSGET. Next the adverbs are scanned to see if the ONPRINTS adverb is included. If so, the PRINON switch is set to true. Now the subroutine branches based on the verb and retrieves and executes the overlay for DESIGN, ALTER, or PRINCE. If DESIGN or ALTER are called, DSPMAK is called when they return.

Subroutine ENTMOD (REPORT) is illustrated in figure 83.

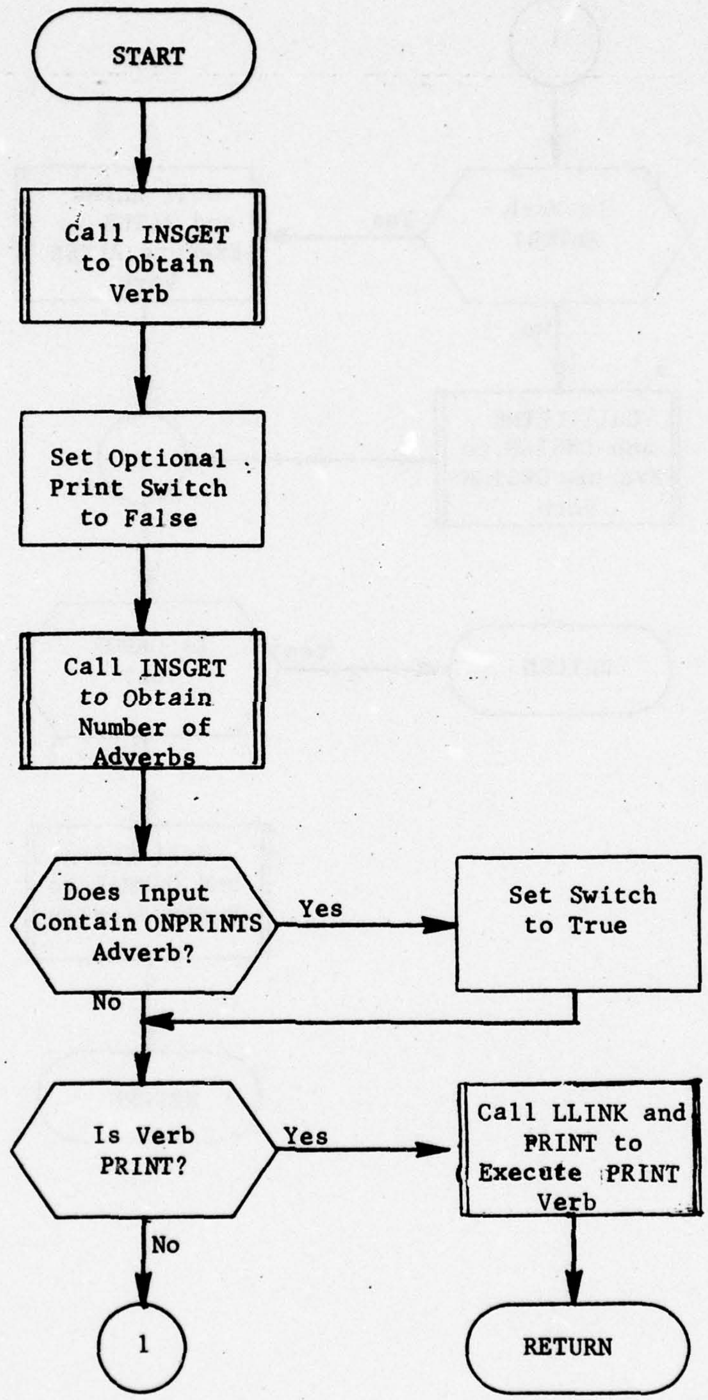


Figure 83. Subroutine ENTMOD (REPORT) (Part 1 of 2)

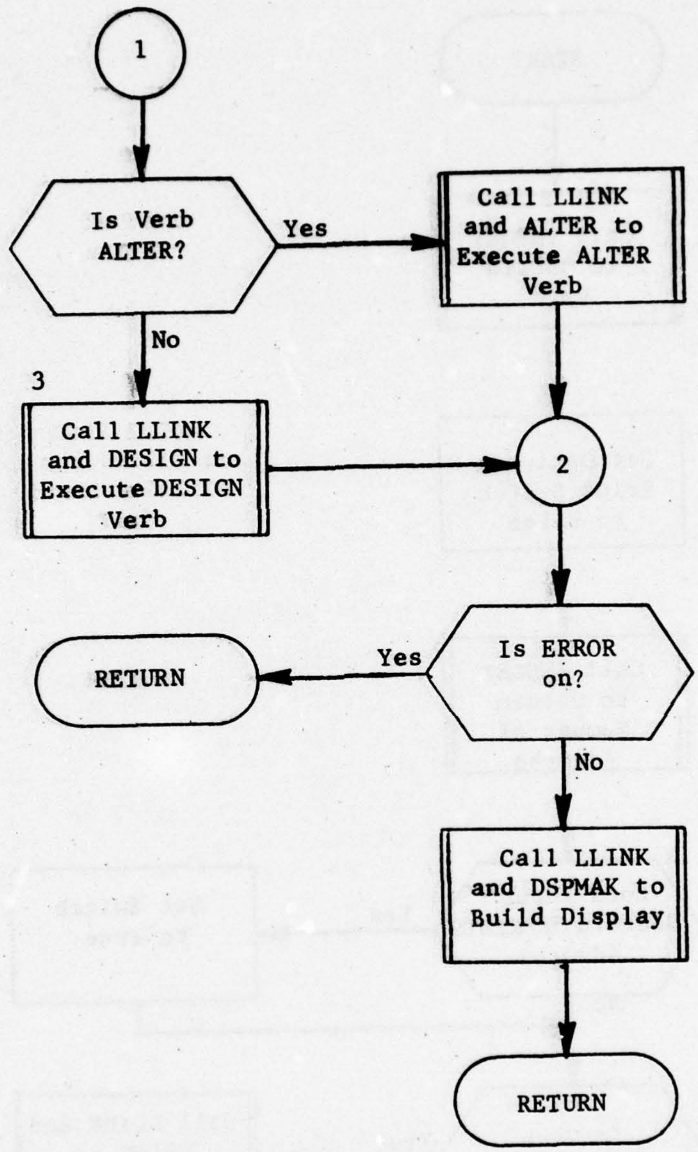


Figure 83. (Part 2 of 2)

6.7.1 Subroutine DSPPUT

PURPOSE: Build and retrieve a display table

ENTRY POINTS: DSPGET, DSPPUT

FORMAL PARAMETERS: ARRAY: Input or output array
LENGTH: Length of array, in words

COMMON BLOCKS: C50, IDPT

SUBROUTINES CALLED: NEXTTT, STORE

CALLED BY: ALTER, DSPMAK, PRINCE

Method:

The method differs depending upon the entry point. Both entry points, however, use IDISPT (common block IDPT) to determine their start point. Therefore, this variable should be set to zero prior to the first call to DSPPUT and to 100 prior to the first call to DSPGET.

Entry DSPPUT

The words of ARRAY are moved one at a time into the input buffer MAIN until the number LENGTH has been moved. Prior to each individual move, IDISPT is incremented. If this indicates the buffer is full, STORE is called and IDISPT reset to one.

Entry DSPGET

First the value of IDISPT is checked. If it is negative, the first word of the retrieval comes from BACKSV. ARRAY is filled from successive positions of MAIN with IDISPT being incremented each time. When IDISPT indicates the buffer has been exhausted, BACKSV is set equal to the last word of the buffer, NEXTTT is called for the next set of words and IDISPT is set to 1.

Subroutine DSPPUT is illustrated in figure 84.

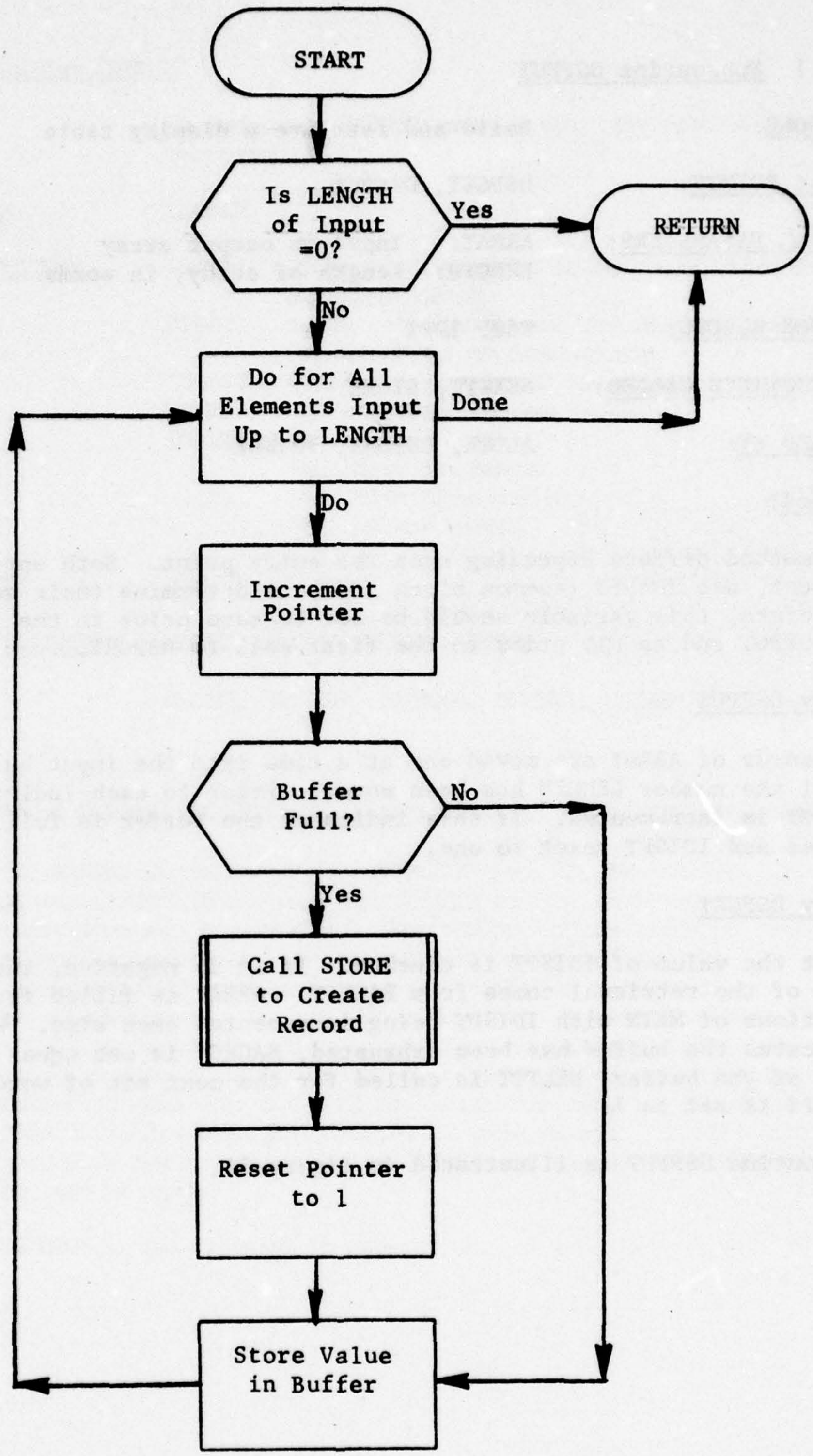


Figure 84. Subroutine DSPPUT: Entry DSPPUT (Part 1 of 2)

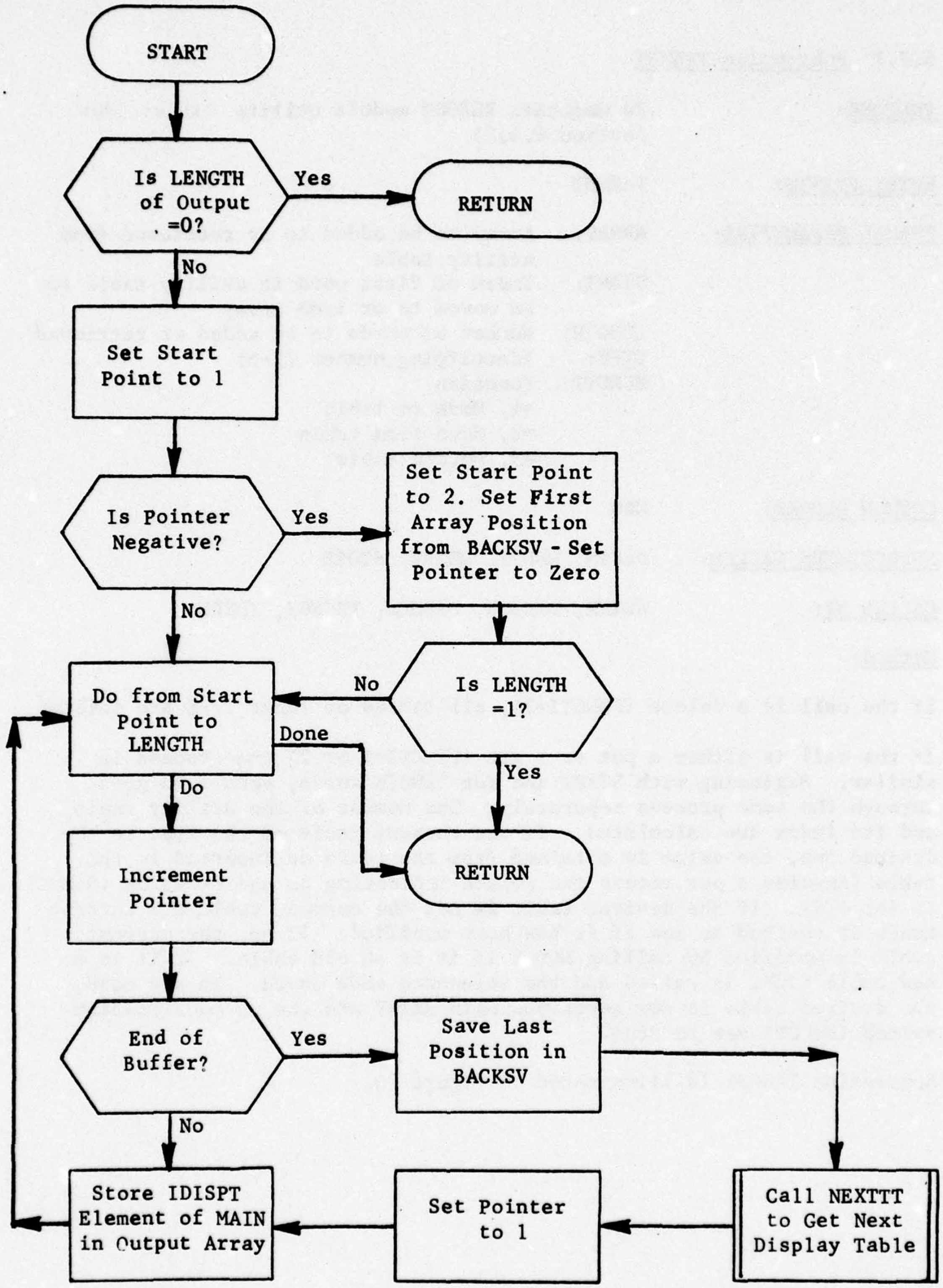


Figure 84. Entry DSPGET (Part 2 of 2)

6.7.2 Subroutine TABMNT

PURPOSE:	To maintain REPORT module utility tables (see section 6.4.2)
ENTRY POINTS:	TABMNT

FORMAL PARAMETERS:

ARRAY:	Array to be added to or retrieved from utility table
START:	Index of first word in utility table to be moved to or from array
LENGTH:	Number of words to be added or retrieved
TYPE:	Identifying number (1-5)
FUNCTI:	Function =1, Move to table =2, Move from table =3, Delete table

COMMON BLOCKS:	C40
SUBROUTINES CALLED:	DLETE, MODFY, RETRV, STORE
CALLED BY:	ALTER, DESIGN, DSPMAK, PRINCE, XDERN

Method:

If the call is a delete (FUNCTI=3), all tables of input TYPE are deleted.

If the call is either a put or a get (FUNCTI=1 or 2) the process is similar. Beginning with START and for LENGTH words, each word goes through the same process separately. The number of the utility table and its index are calculated. If the current table of the type is the desired one, the value is obtained from the table or inserted in the table (anytime a put occurs the switch indicating no modification (NOMOD) is set off). If the desired table is not the current table, the current table is checked to see if it has been modified. If so, the current table is modified by calling MODFY if it is an old table. If it is a new table STORE is called and the reference code saved. In any case, the desired table is now retrieved with RETRV and the no modification switch (NOMOD) set to true.

Subroutine TABMNT is illustrated in figure 85.

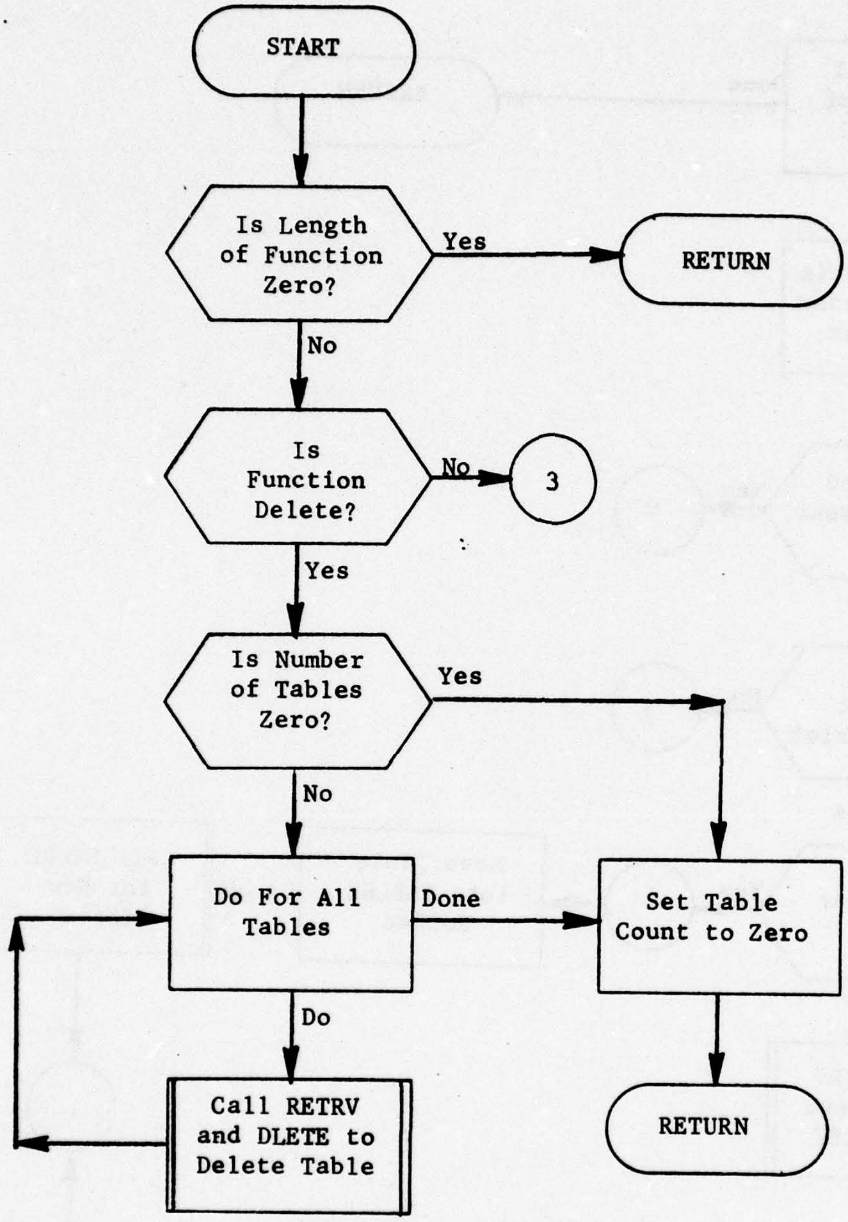


Figure 85. Subroutine TABMNT (Part 1 of 3)

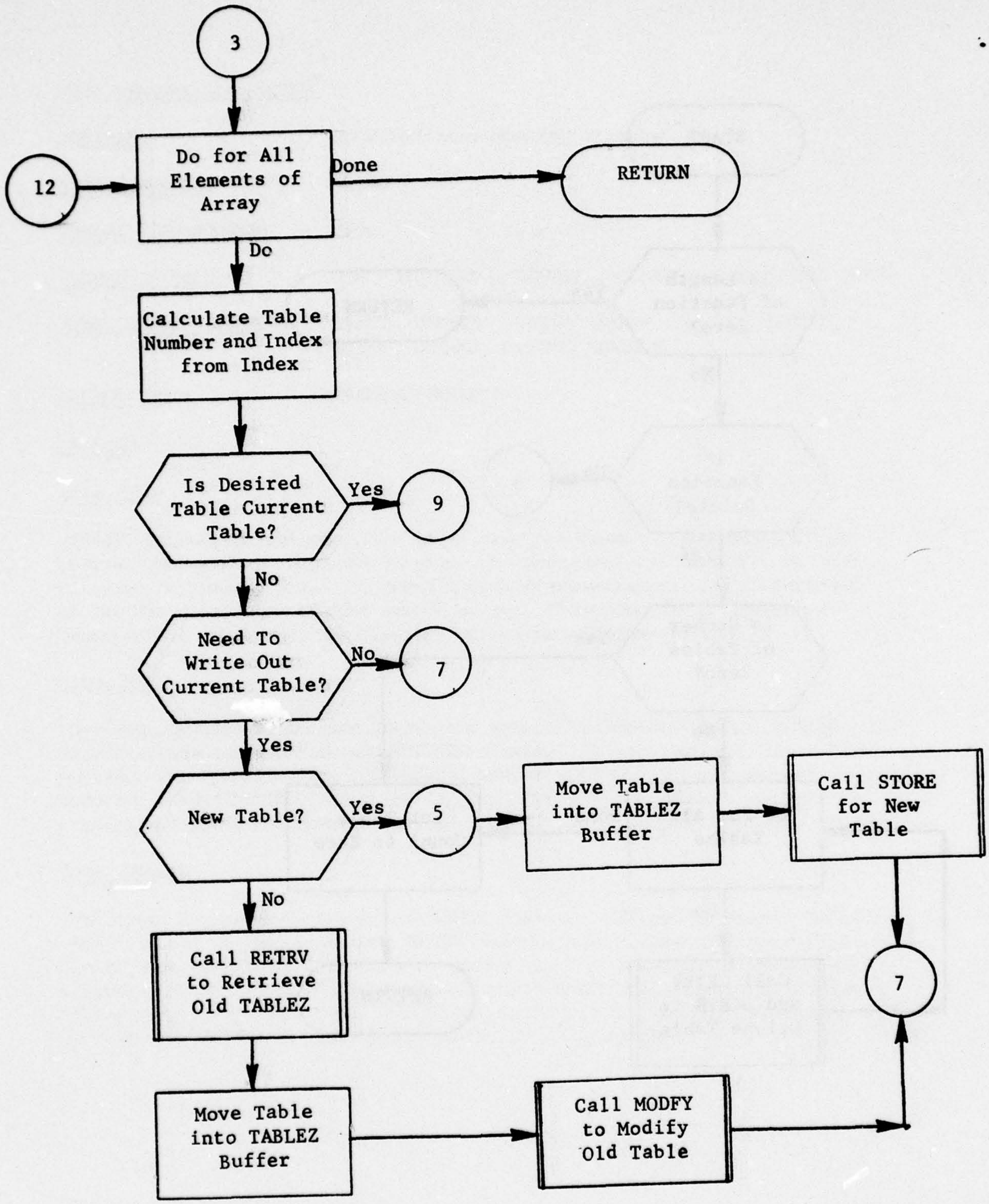


Figure 85. (Part 2 of 3)

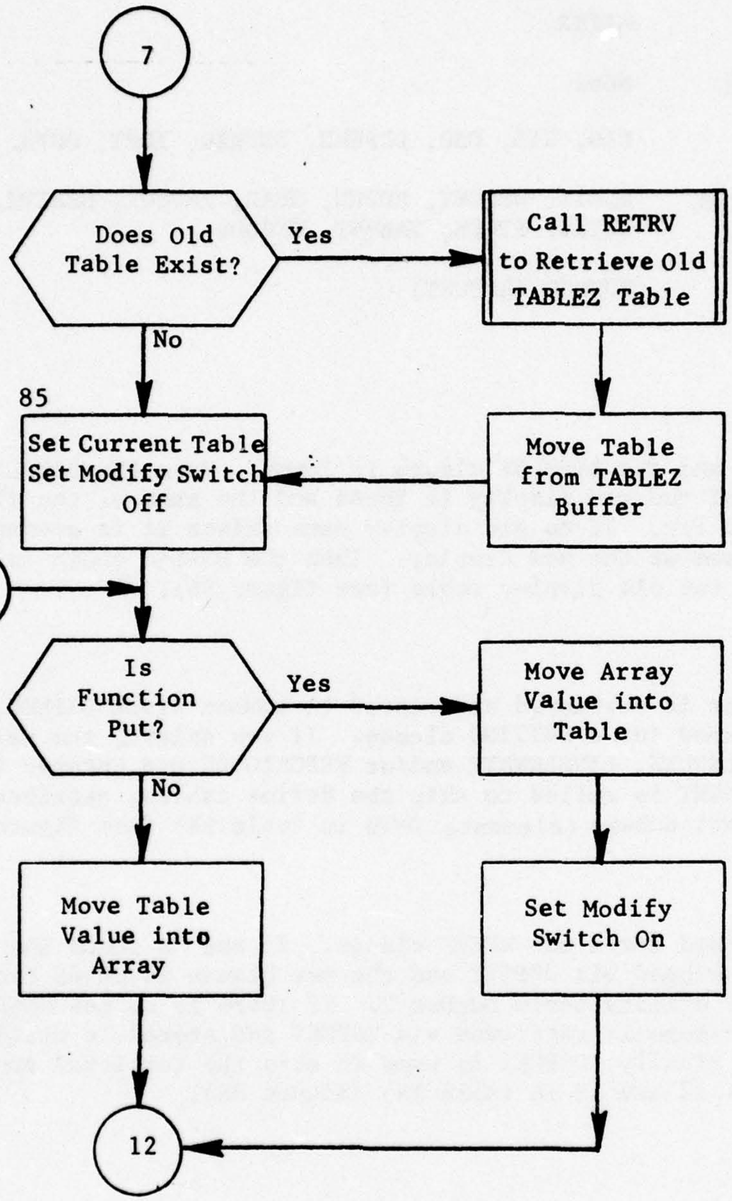


Figure 85. (Part 3 of 3)

6.8 Subroutine ALTER*

PURPOSE:	To alter an existing display
ENTRY POINTS:	ALTER
FORMAL PARAMETERS:	None
COMMON BLOCKS:	C10, C15, C30, DEFNMZ, DSPHED, IDPT, OOPS, ZEES
SUBROUTINES CALLED:	DLETE, DSPGET, HDFND, HEAD, INSGET, NEXTTT, RETRV, STORE, TABMNT, UNCODE
CALLED BY:	ENTMOD (REPORT)

Method:

Step One

INSGET is called and the DISPLAY clause is found. From the DISPLAY clause the name of the new display is found and the name of the old display is looked for. If no old display name exists it is assumed to be the same name as the new display. Then the DSPLAY chain is searched to find the old display table (see figure 86).

Step Two

The display header is retrieved and stored in common block DSPHED. The input is now scanned for a SETTING clause. If one exists, the values entered for PAGELENGTH, LINELENGTH and/or REPORTCODE are entered in the header. Next DSPGET is called to skip the define tables, attribute tables and the sort scheme (elements 2-10 in table 18) (see figure 87).

Step Three

The input is scanned for a new WHERE clause. If one is found the old WHERE clause is skipped via DSPGET and the new clause is moved from the input into TABMNT utility table number 2. If there is no new WHERE clause, the old clause is retrieved via DSPGET and stored in utility table number 2. Finally, DSPGET is used to skip the retrieval and print schemes (elements 12 and 13 in table 18) (figure 88).

*Main routine of overlay link RPTALT

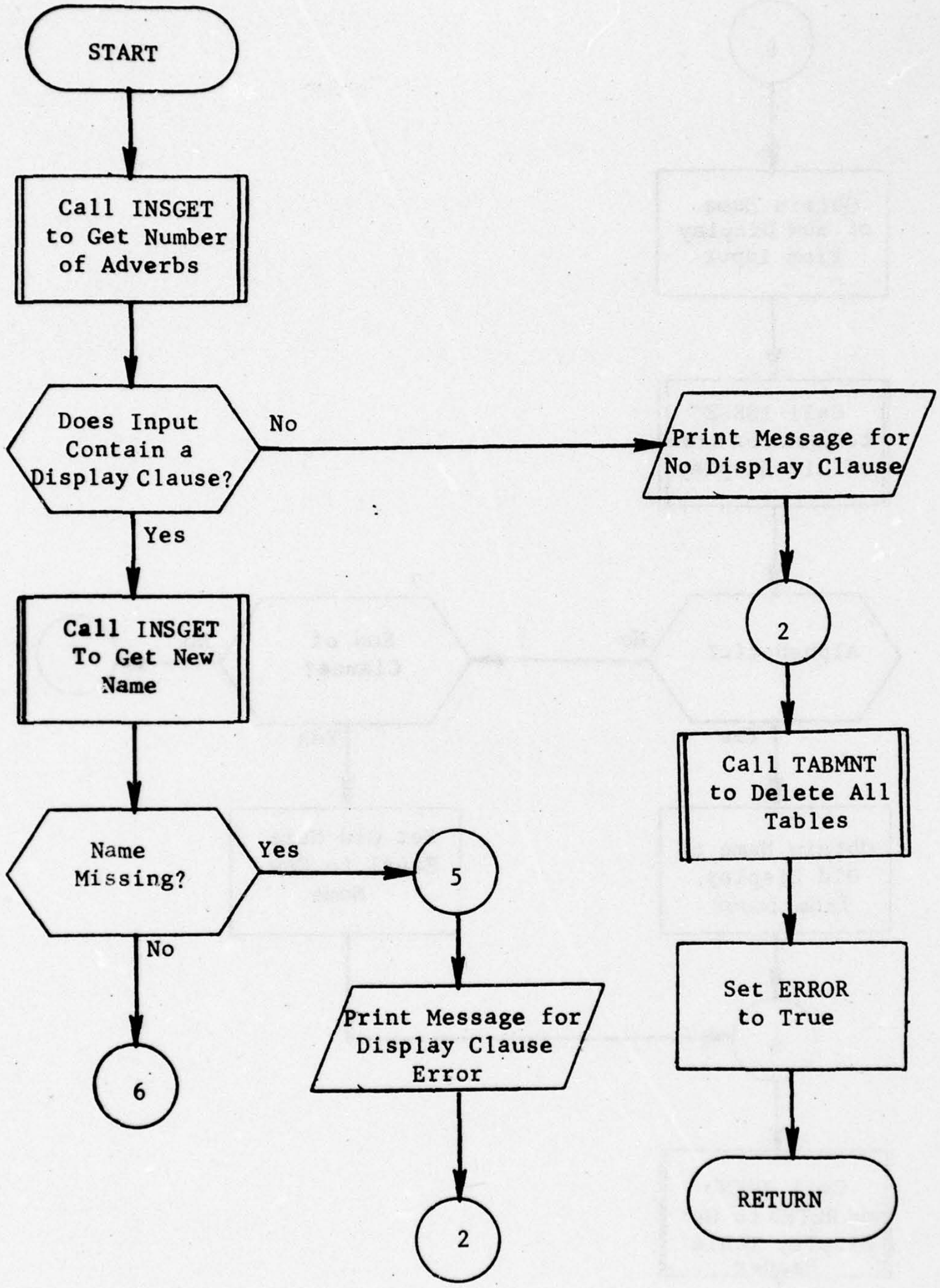


Figure 86. Subroutine ALTER: Step One (Part 1 of 3)

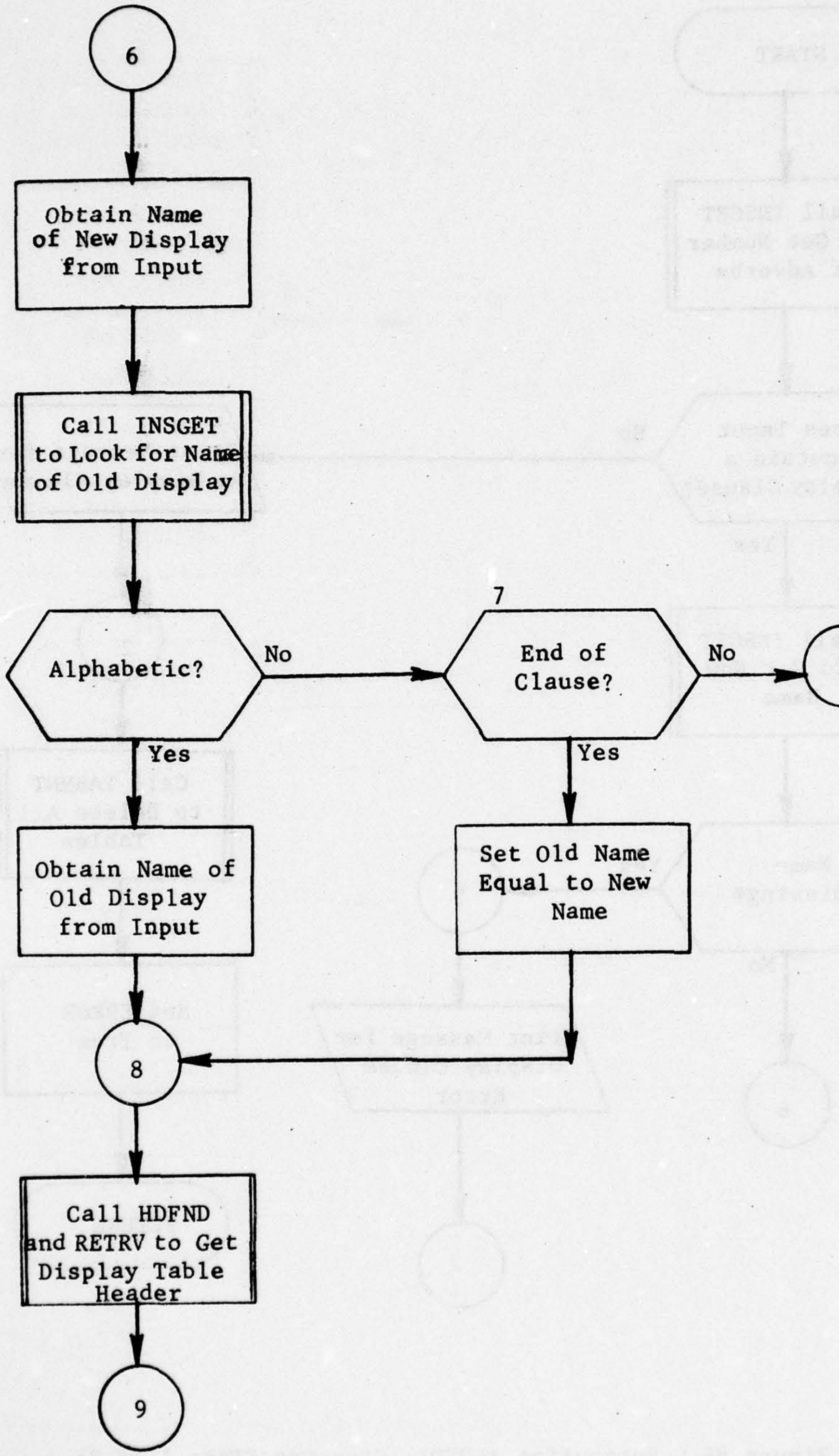


Figure 86. (Part 2 of 3)

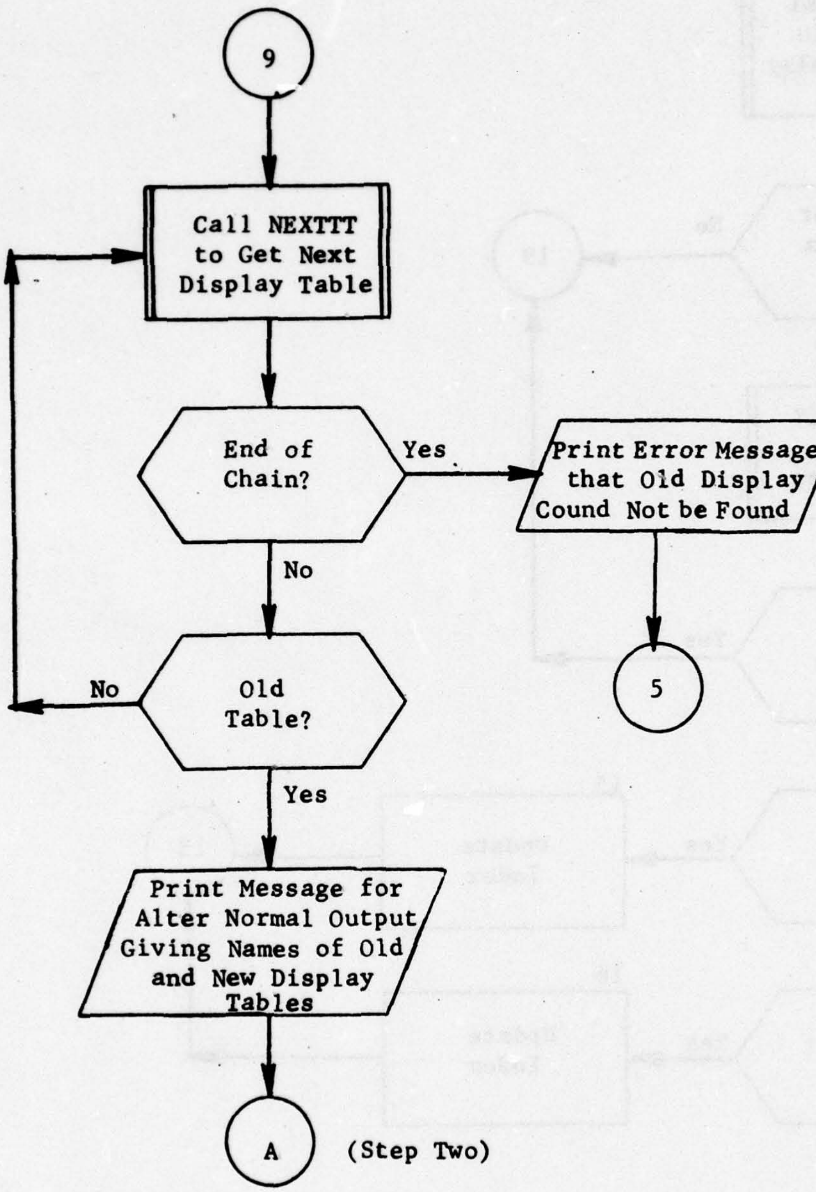


Figure 86. (Part 3 of 3)

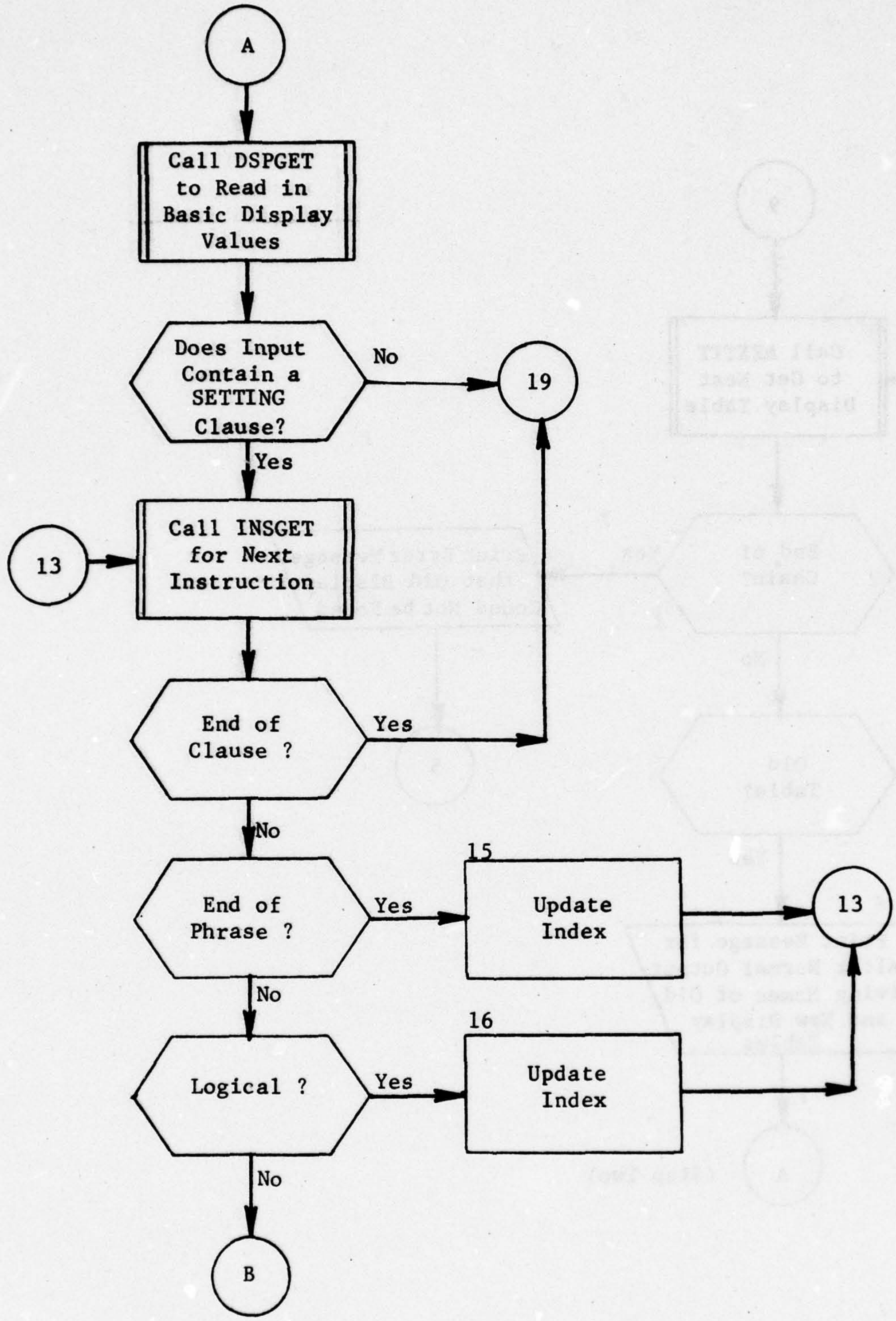


Figure 87. Subroutine ALTER: Step Two (Part 1 of 3)

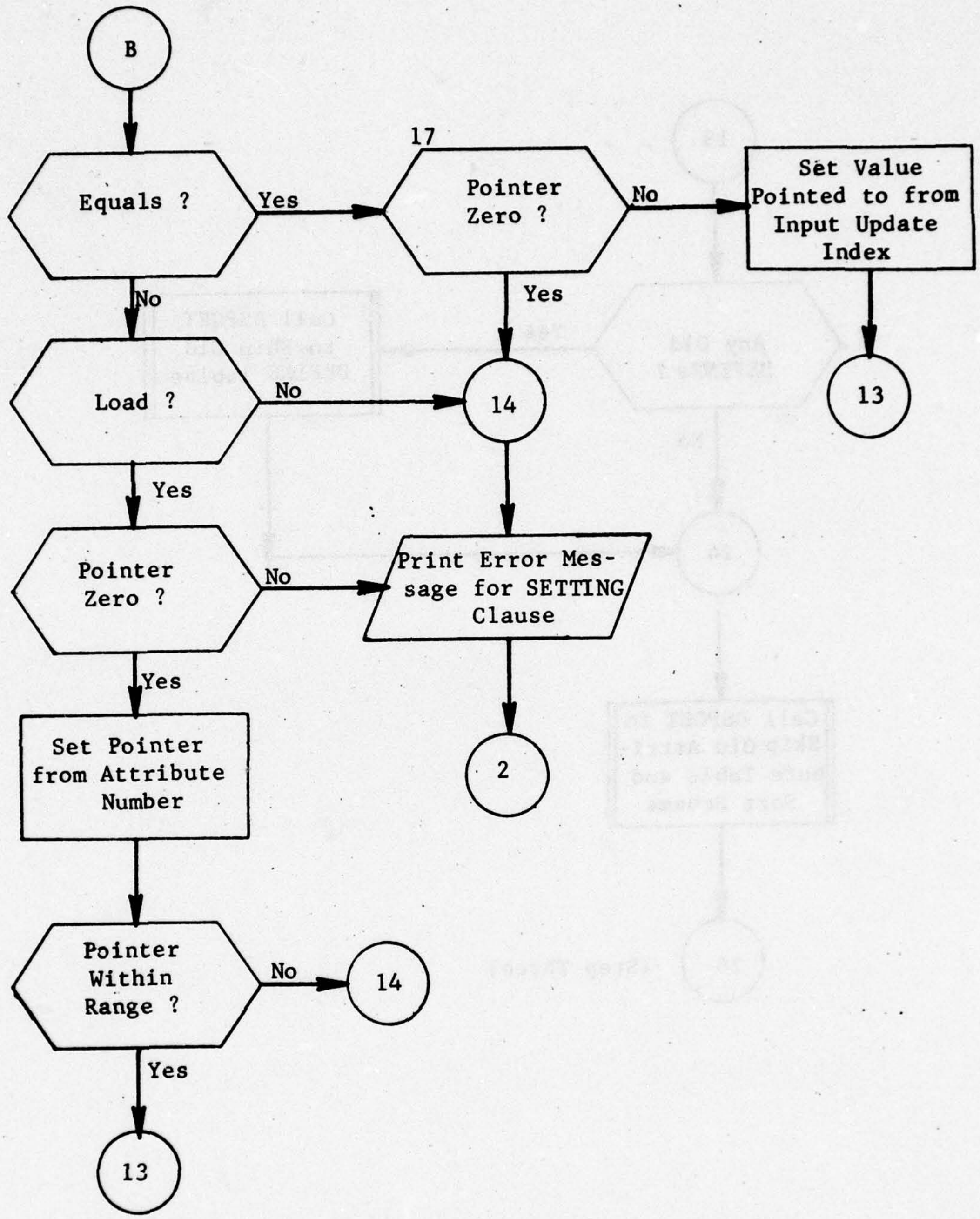


Figure 87. (Part 2 of 3)

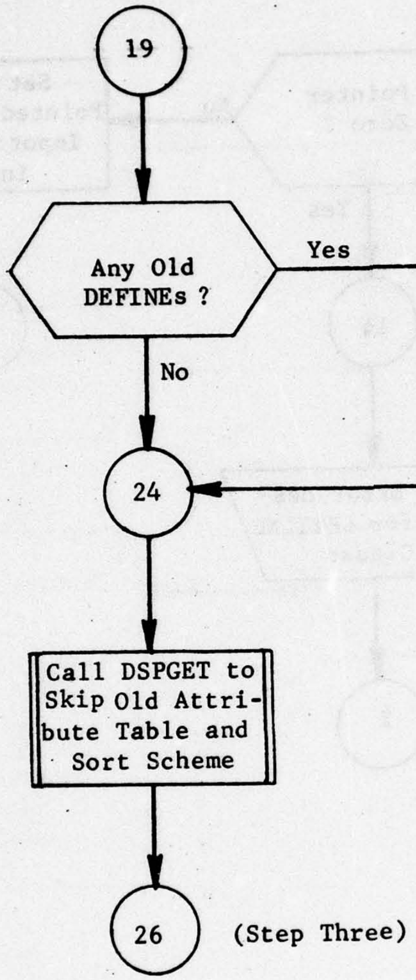


Figure 87. (Part 3 of 3)

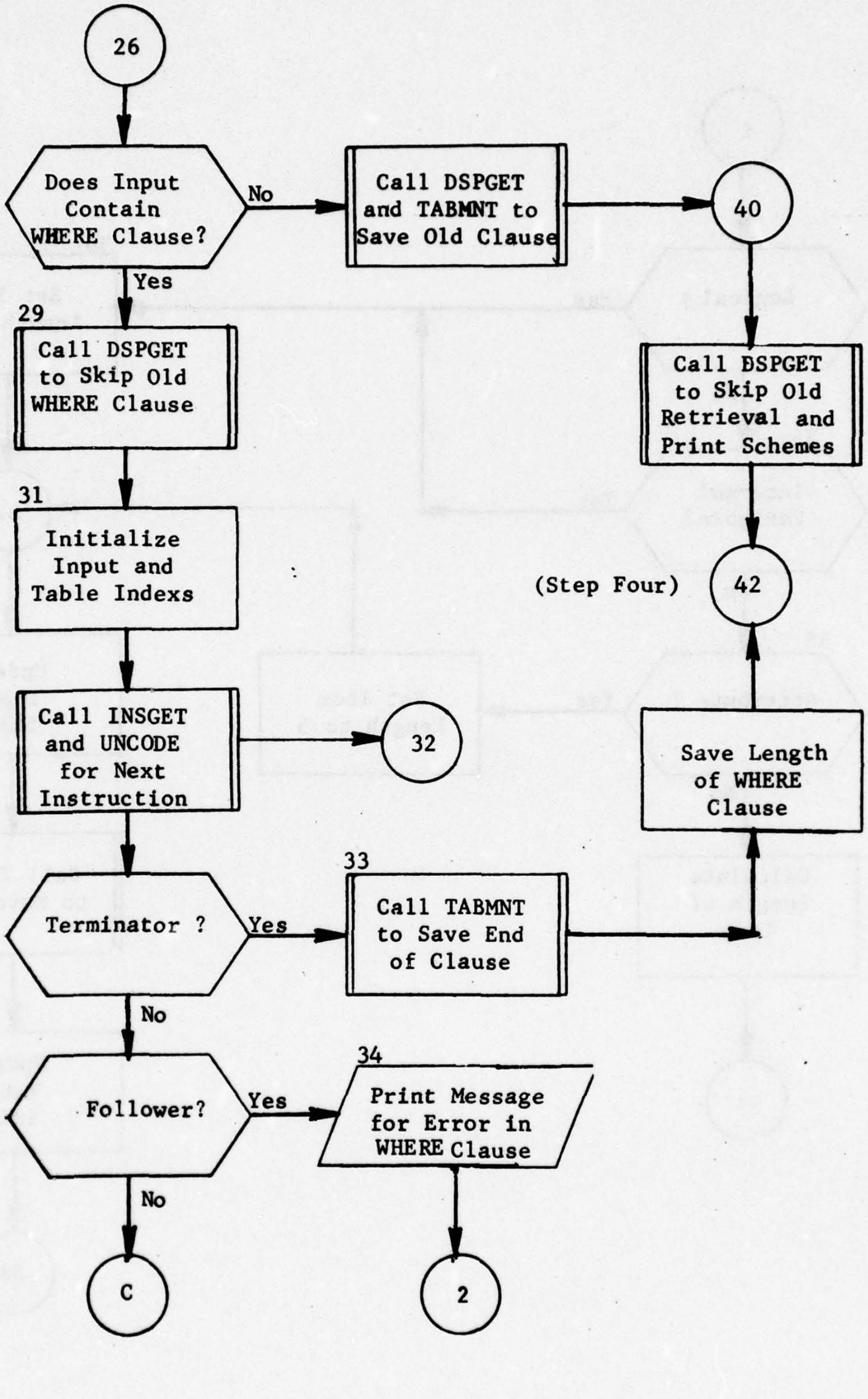


Figure 88. Subroutine ALTER: Step Three (Part 1 of 2)

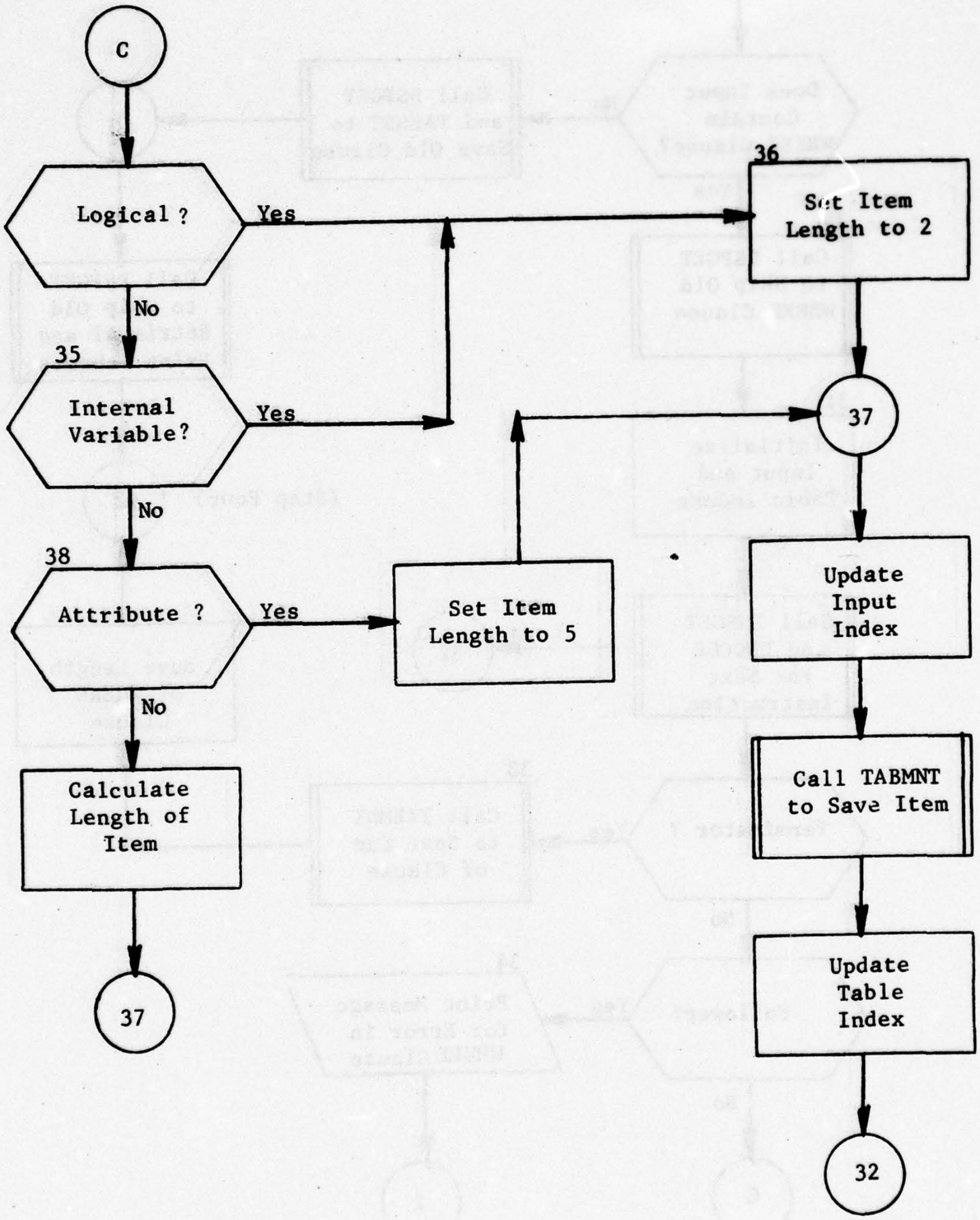


Figure 88. (Part 2 of 2)

Step Four

Any new DEFINE clauses are read in and their names saved in common block DEFNMZ. Their instructions are stored in utility table 1. If the new DEFINE is of the form DEFINE name=DEFINE name, it is not saved as above but rather stored in the delete list (DELDEL). Next, DSPGET is used to retrieve any old DEFINEs. If the old DEFINEs name is either already in the DEFINE list or in the delete list it is ignored. Otherwise it is saved like a new DEFINE (see figure 89).

Step Five

If a new SORT clause was input, it is read in and stored in utility table 3 and the old SORT clause is skipped. If there is no new SORT clause, the old clause is retrieved via DSPGET and stored in utility table 3 (see figure 90).

Step Six

If there is no new FORMAT clause the old clause is retrieved using DSPGET and stored in utility table 4. Otherwise each AFTER, REMOVE or REPLACE command is executed in the order input. As each command contains an identified PAGE, LINE, HEADER, or TRAILER item, the old clause is retrieved and stored in utility table 4 up to the identified item. If the command is AFTER the identified item is also stored and the items in the input which follow the AFTER command are stored in utility table 4 next. If the command is REMOVE and the item is not PAGE, the identified item is skipped. If the item was PAGE, the REMOVE command causes the identified item and all following items up to the next PAGE to be skipped. If the command is REPLACE the item is skipped. Then, if a non-PAGE item, all items following the REPLACE command are stored in utility table 4.

During this process each individual element (attribute, long string, alphabetic constant, etc.) is examined for validity and added to the printed display report (see figure 91).

Step Seven

If the old and new display are the same the old one is deleted. A new display table record (DISPRC) is now stored (see figure 92).

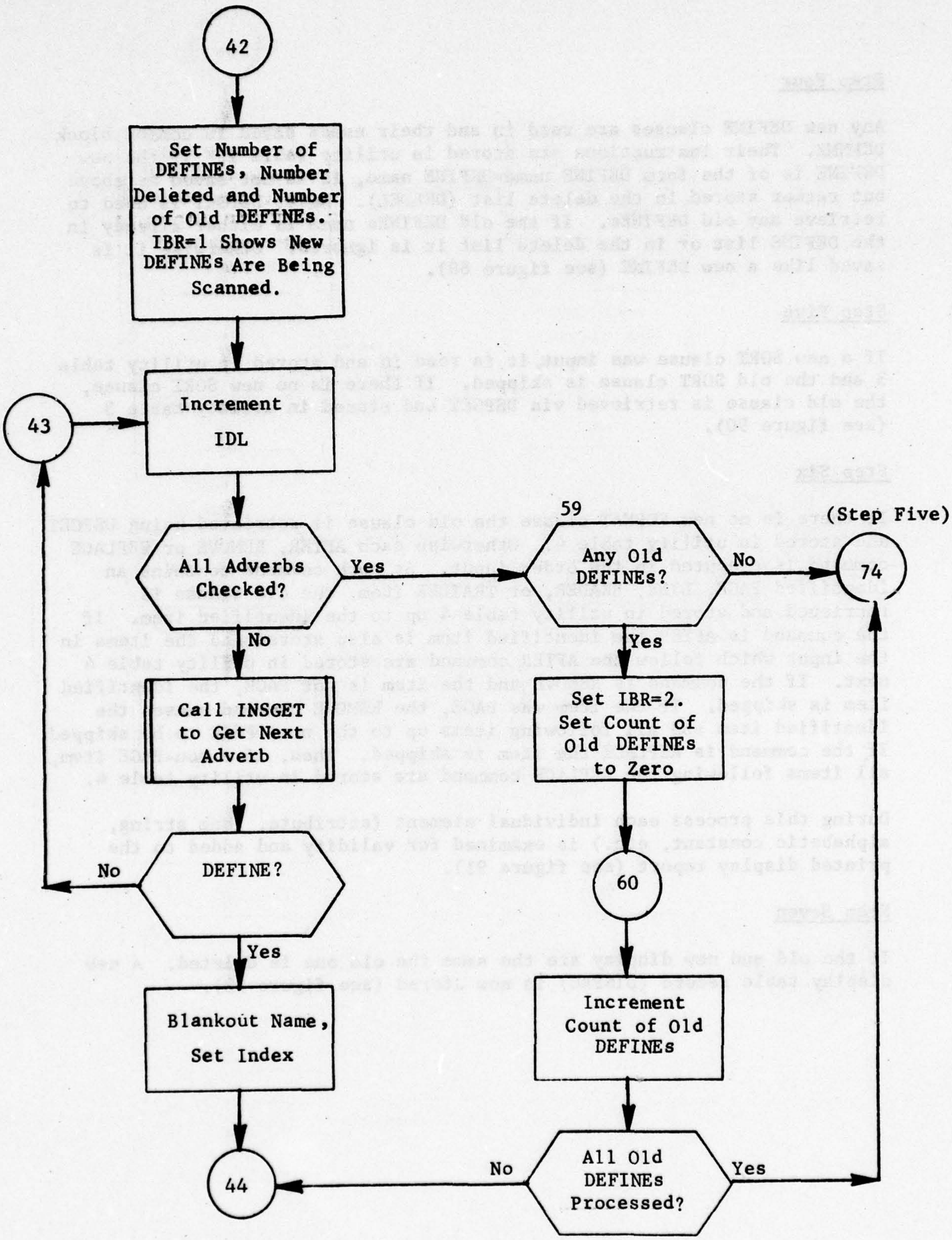


Figure 89. Subroutine ALTER: Step Four (Part 1 of 8)

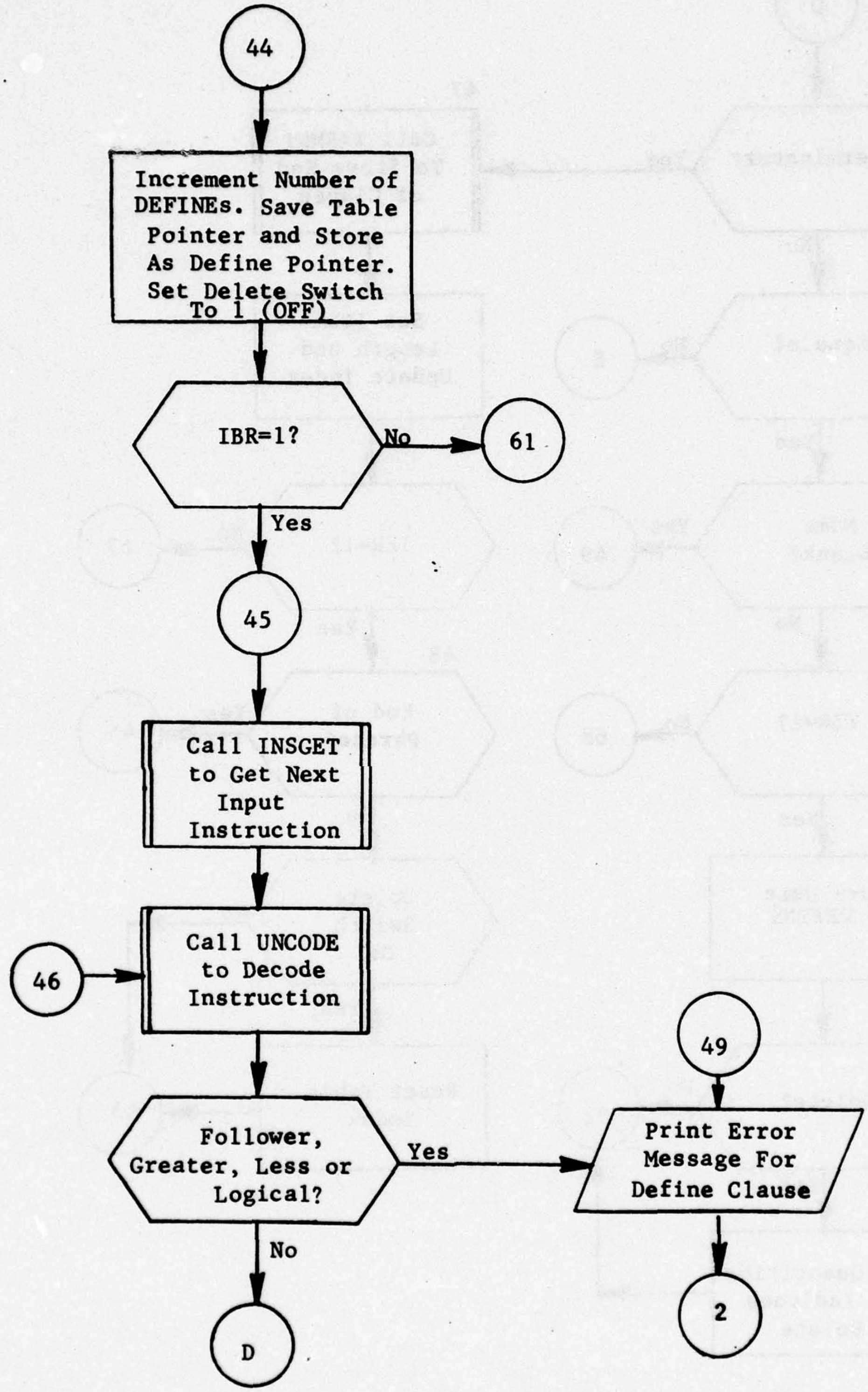


Figure 89. (Part 2 of 8)

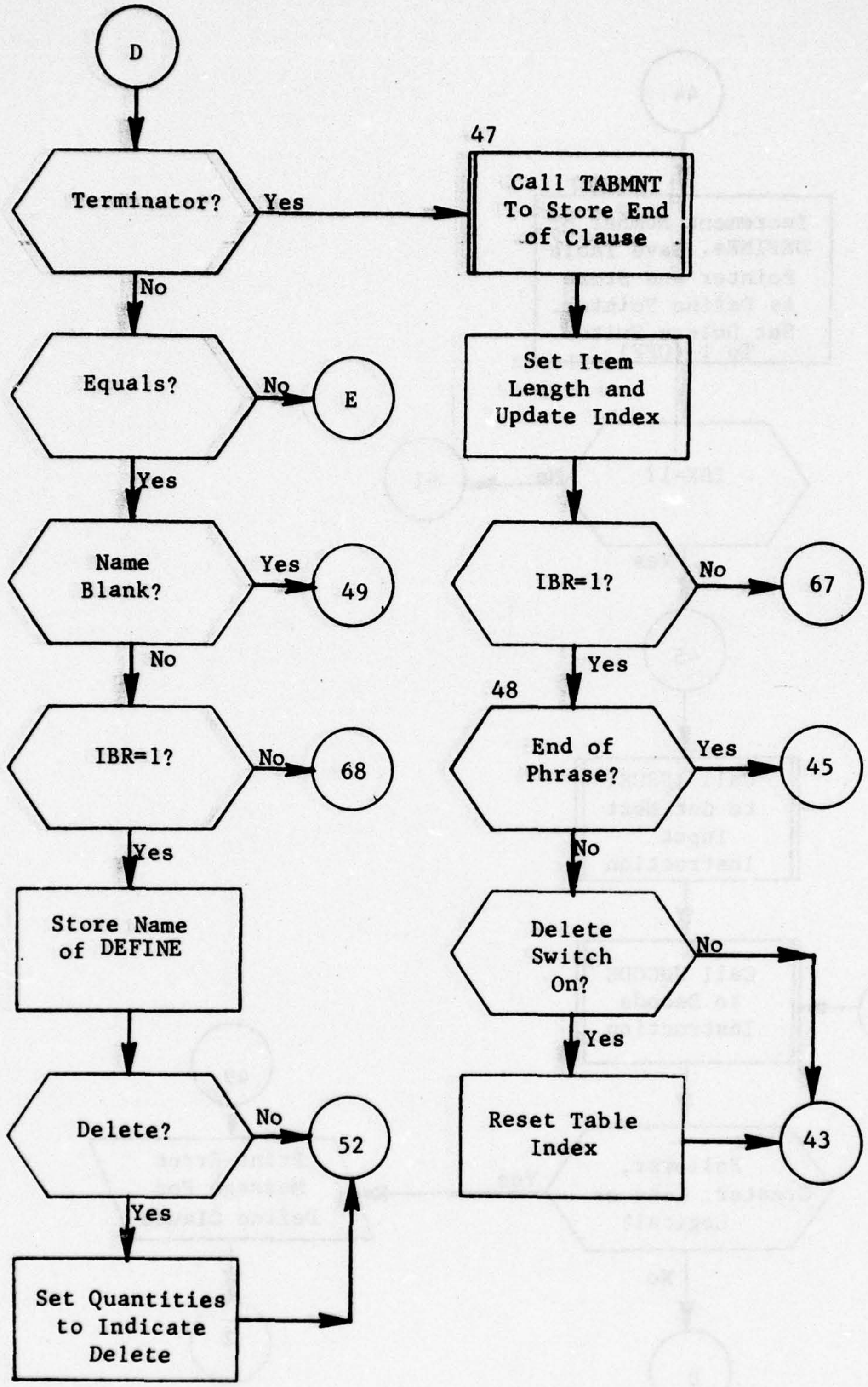


Figure 89. (Part 3 of 8)

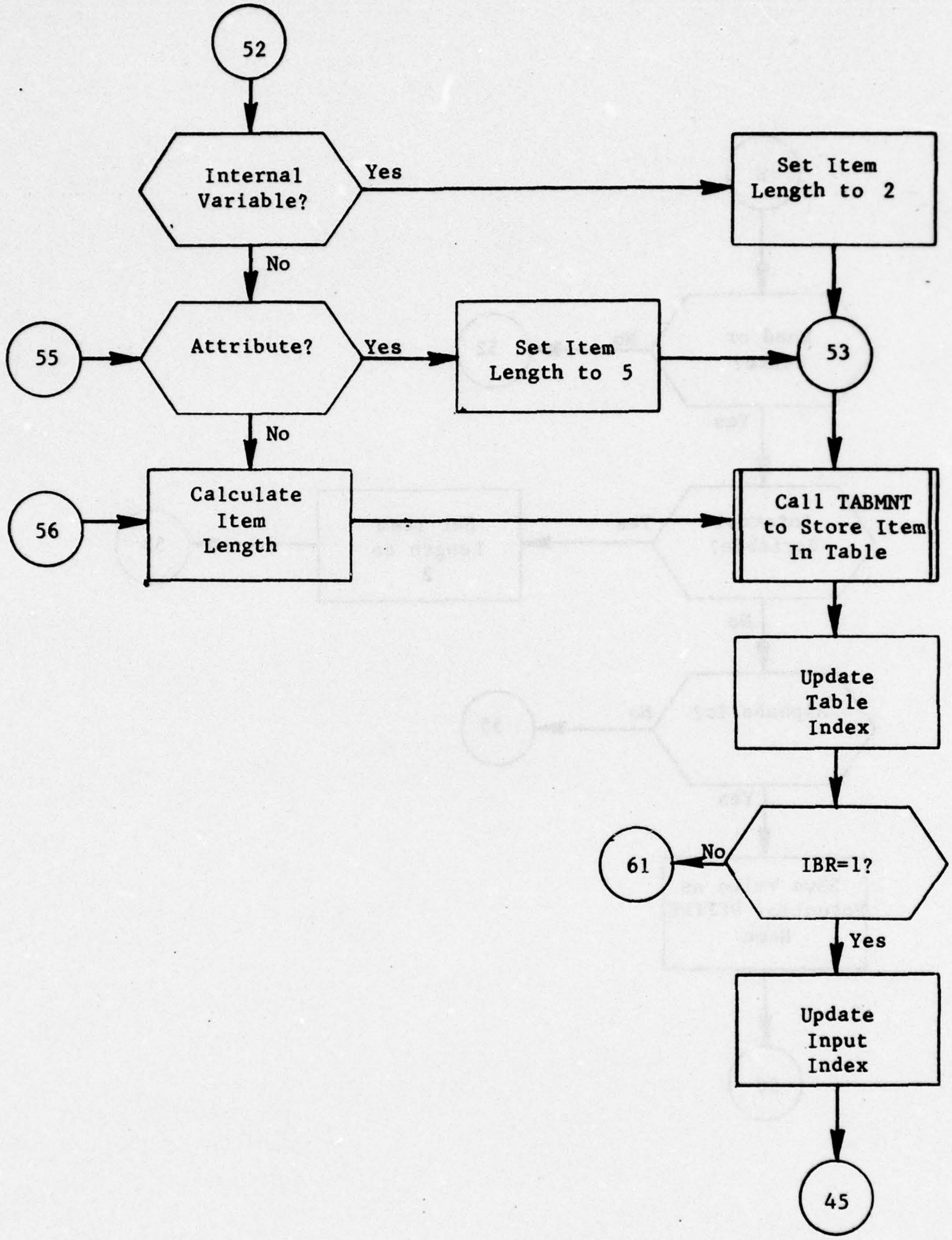


Figure 89. (Part 4 of 8)

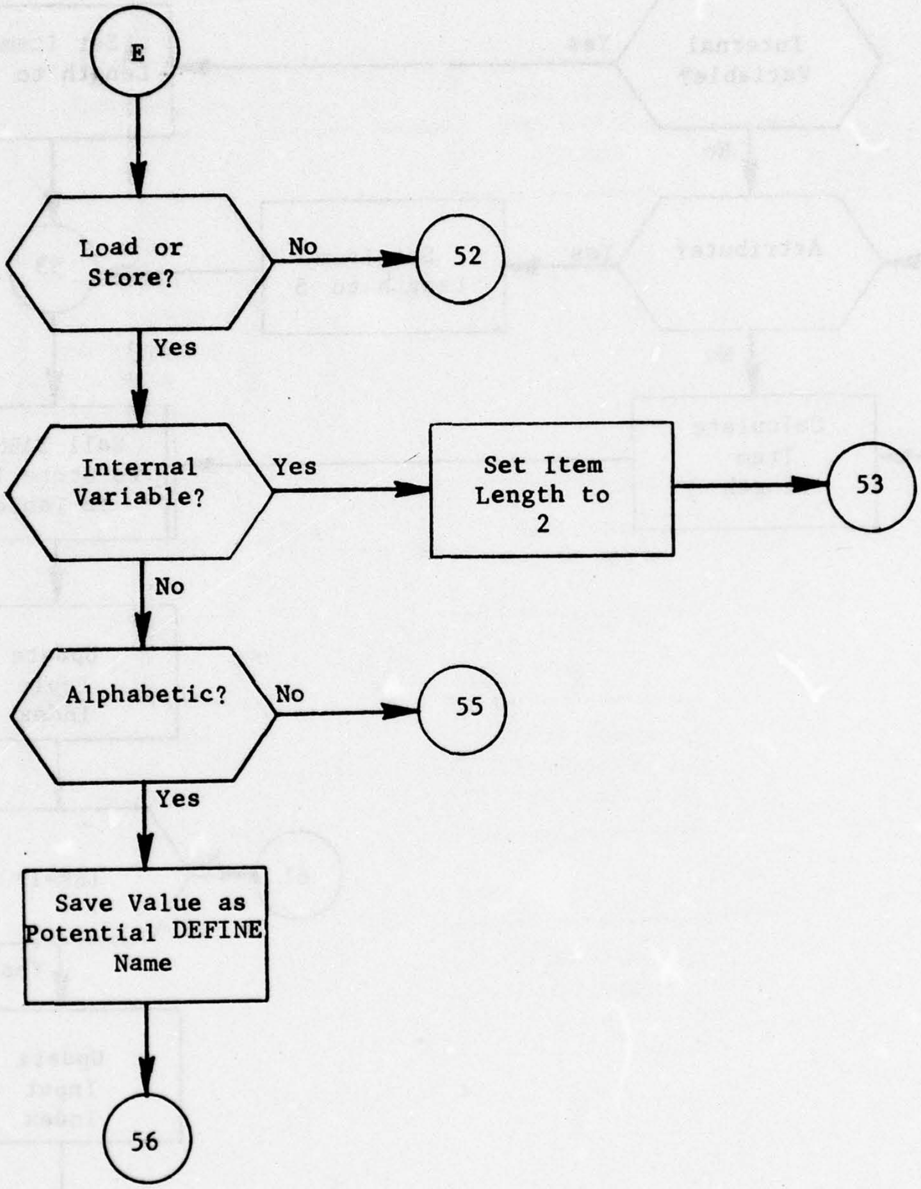


Figure 89. (Part 5 of 8)

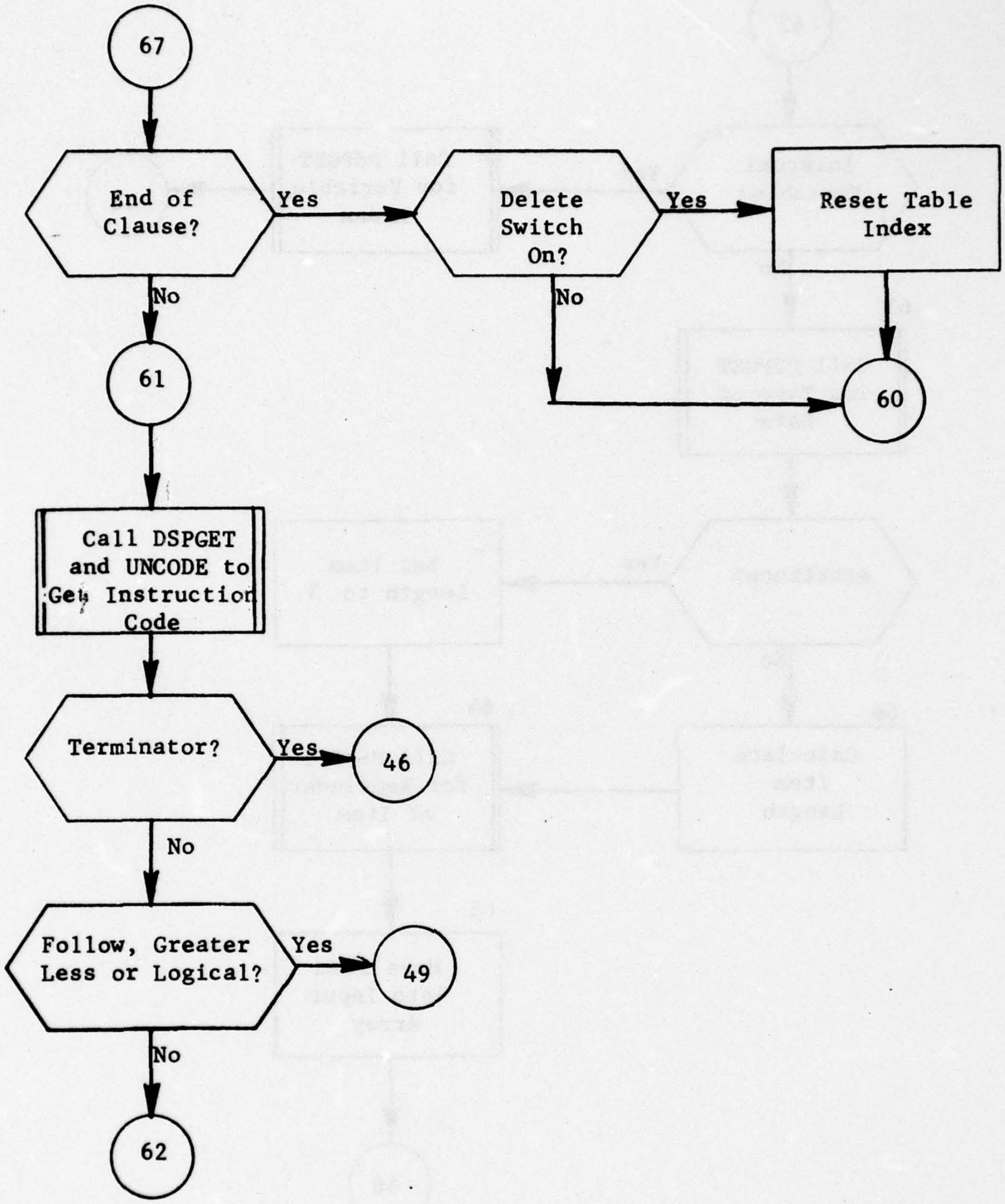


Figure 89. (Part 6 of 8)

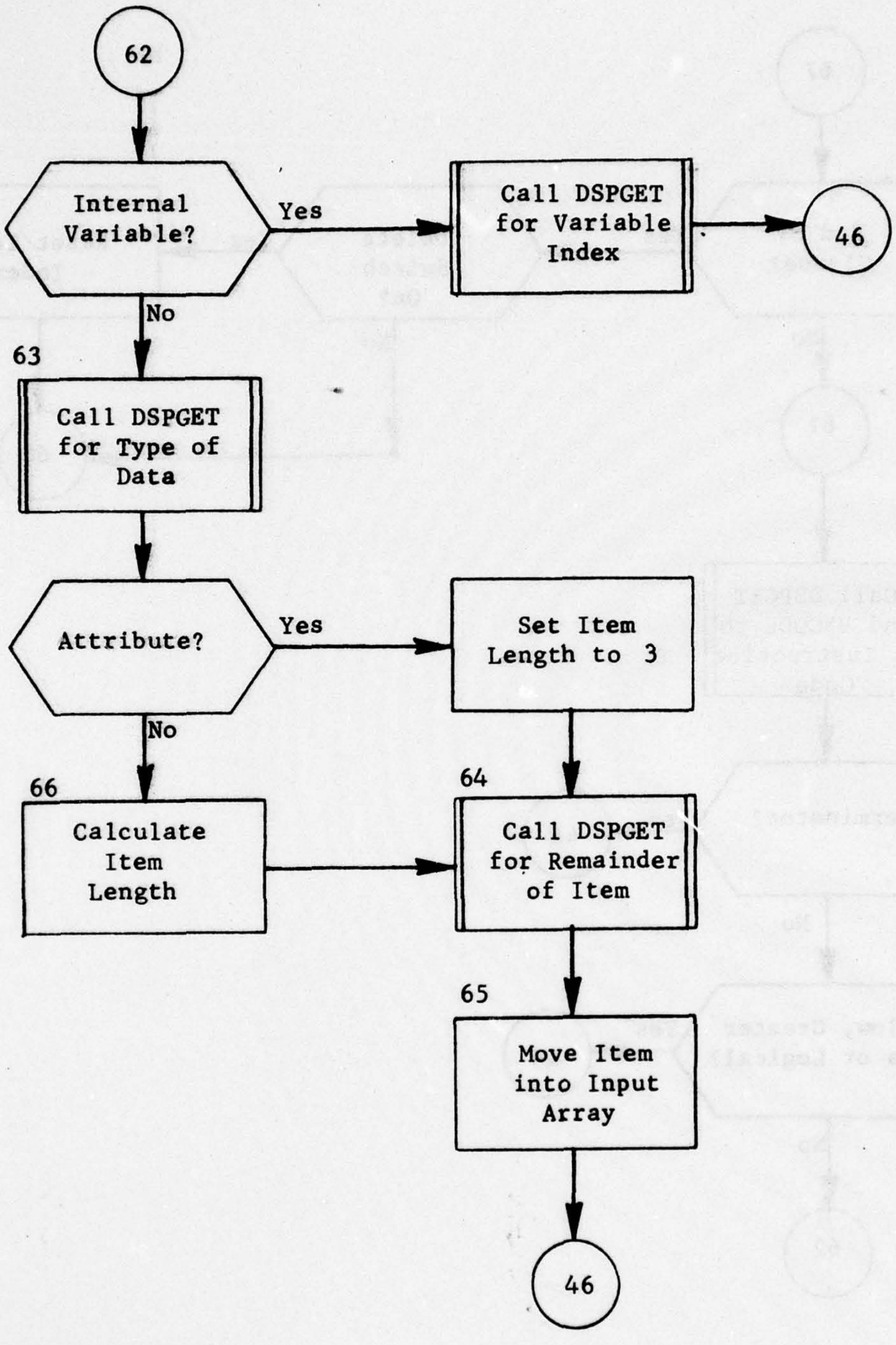


Figure 89. (Part 7 of 8)

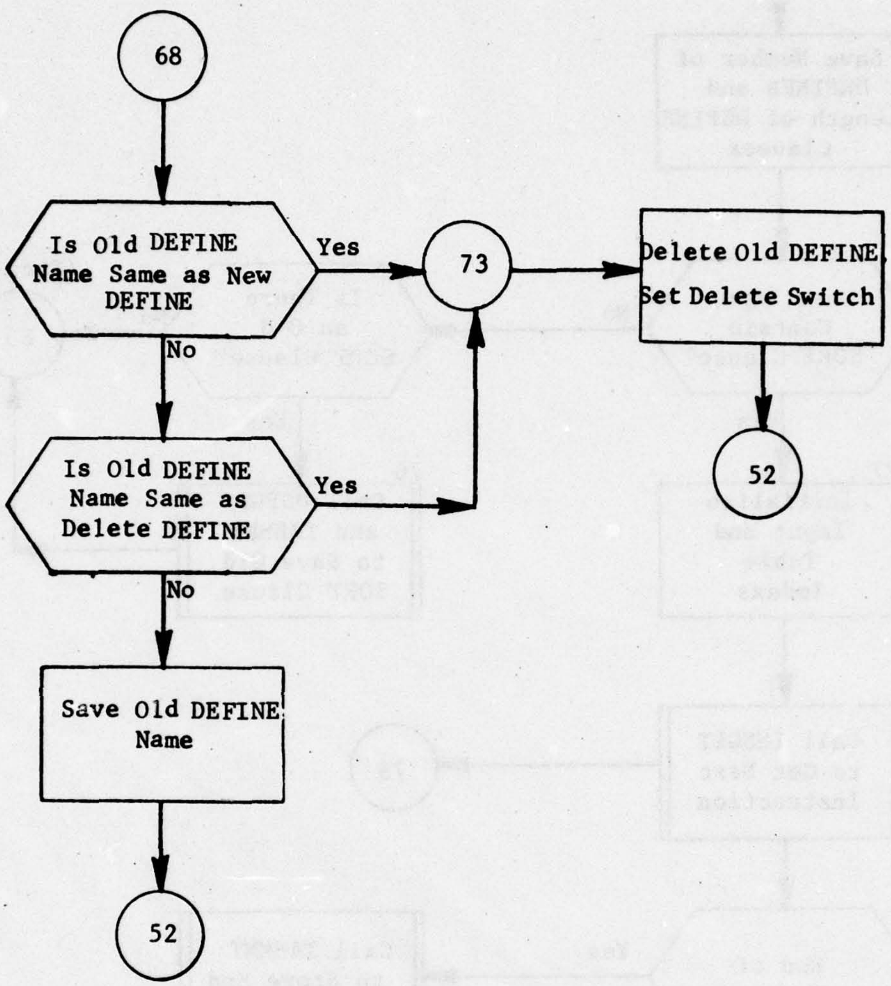


Figure 89. (Part 8 of 8)

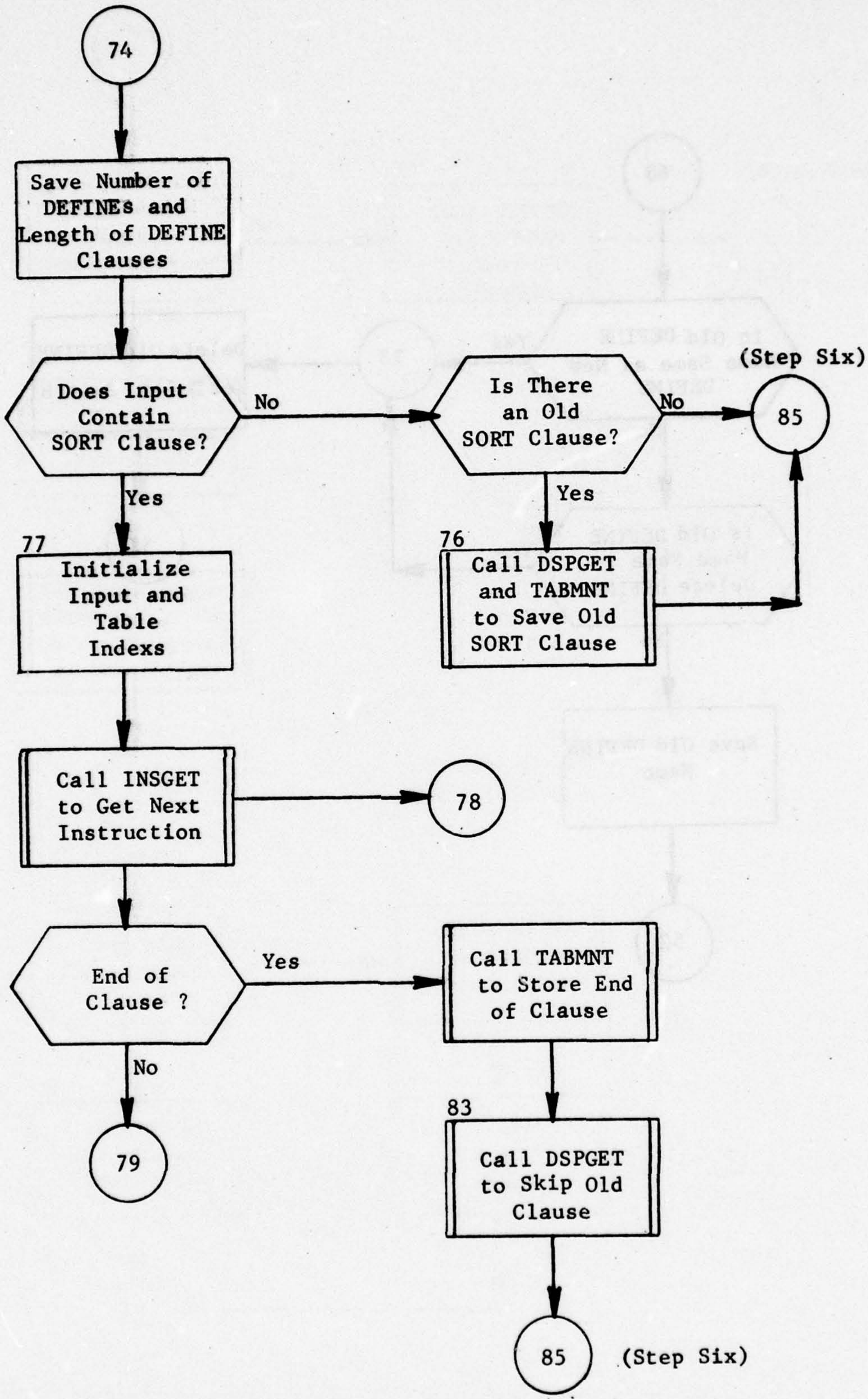


Figure 90. Subroutine ALTER: Step Five (Part 1 of 2)

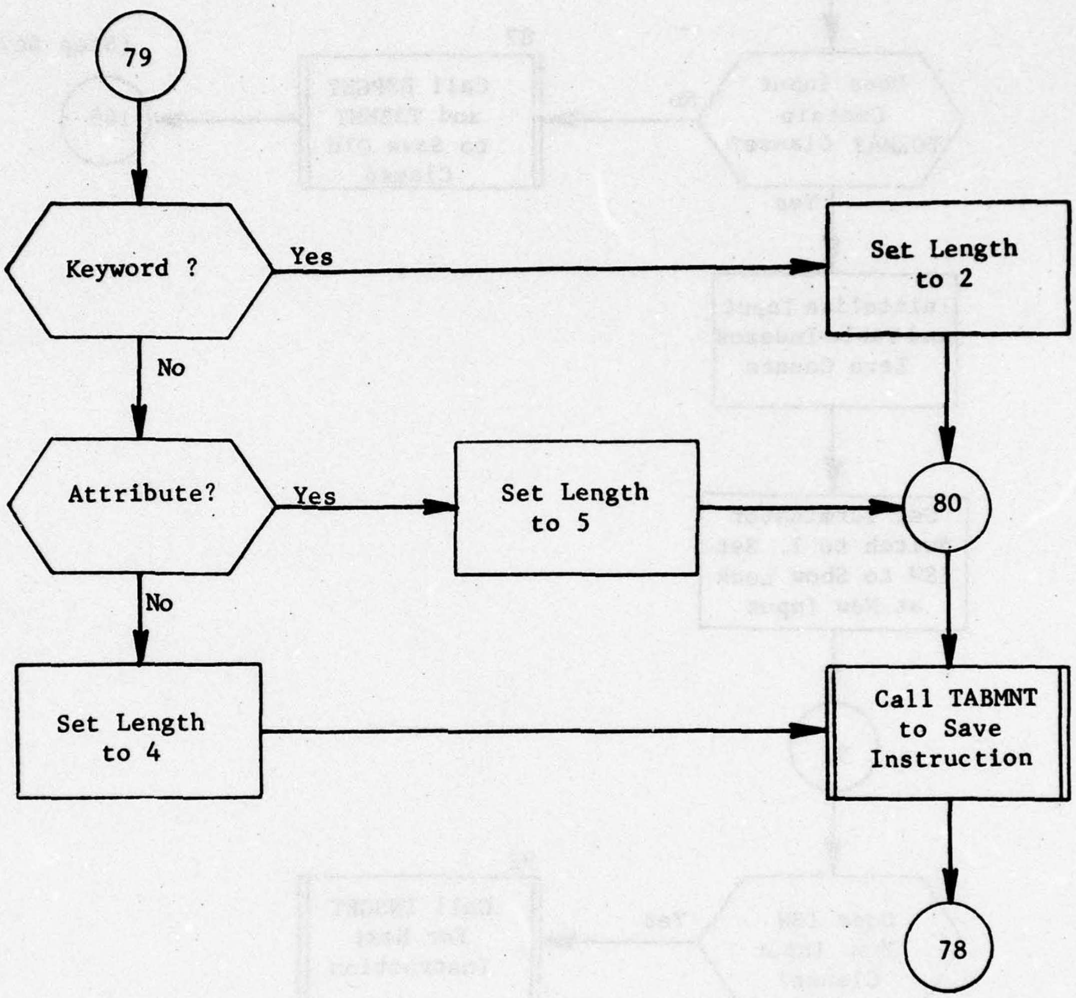


Figure 90. (Part 2 of 2)

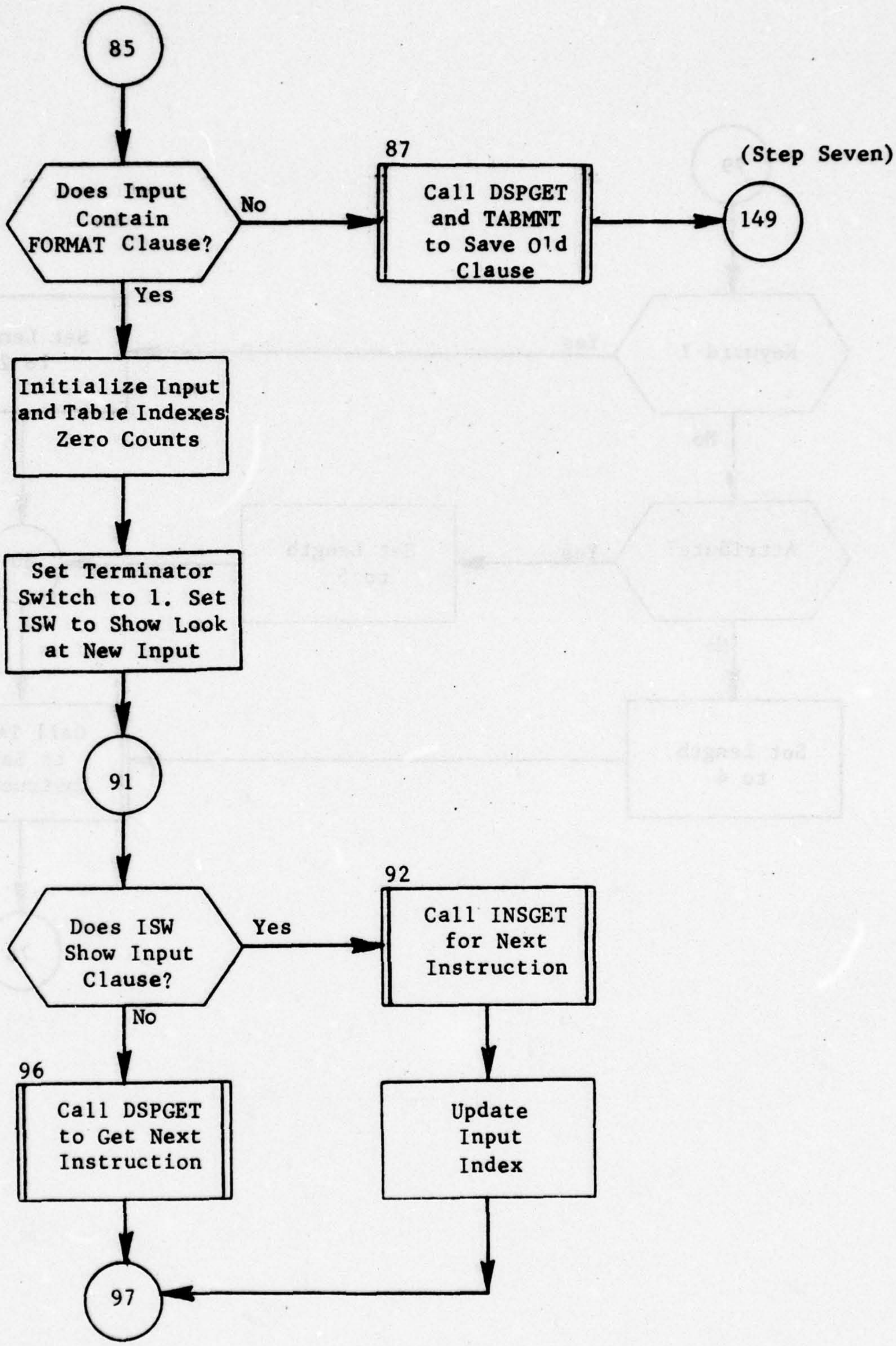


Figure 91. Subroutine ALTER: Step Six (Part 1 of 11)

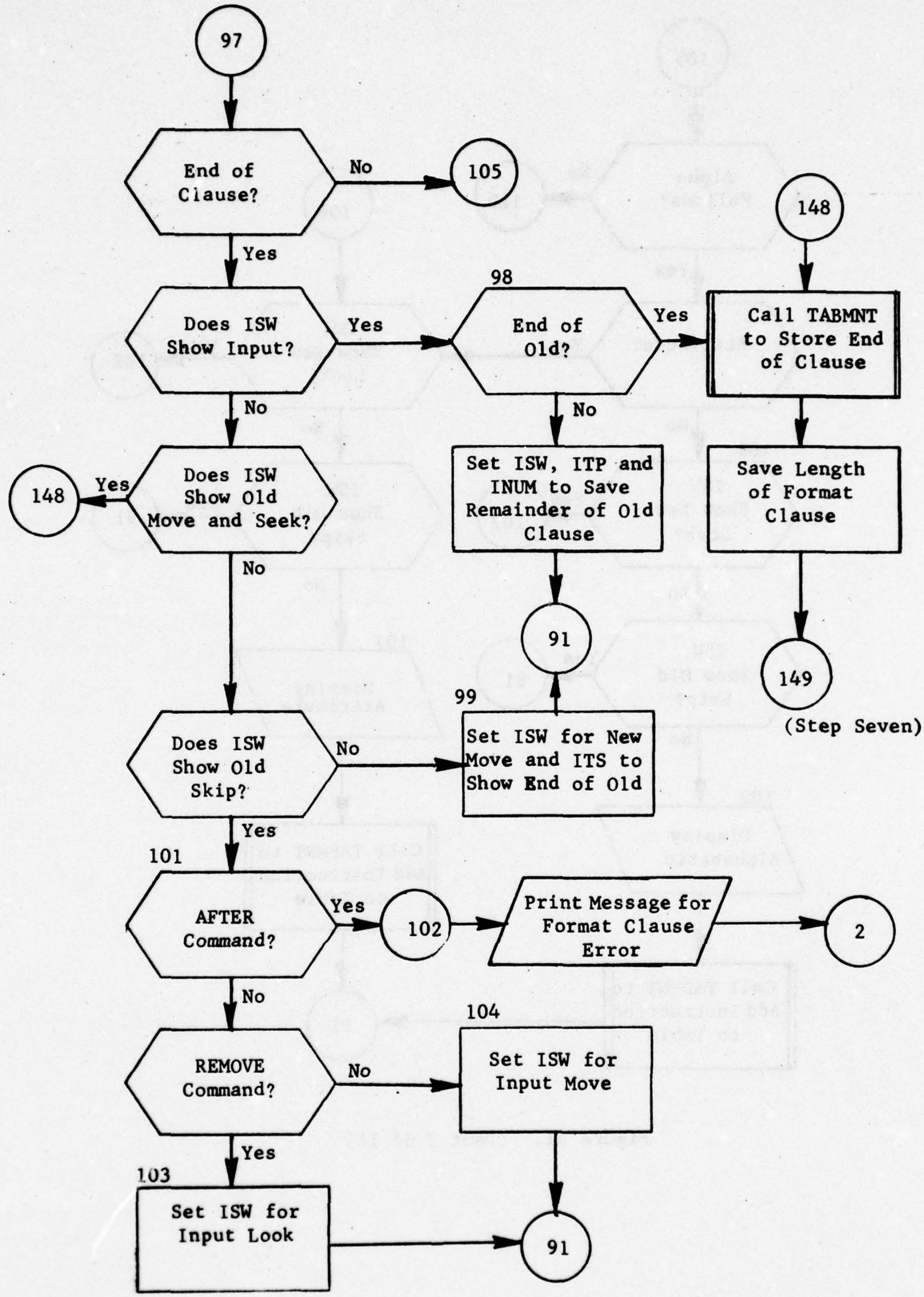


Figure 91. (Part 2 of 11)

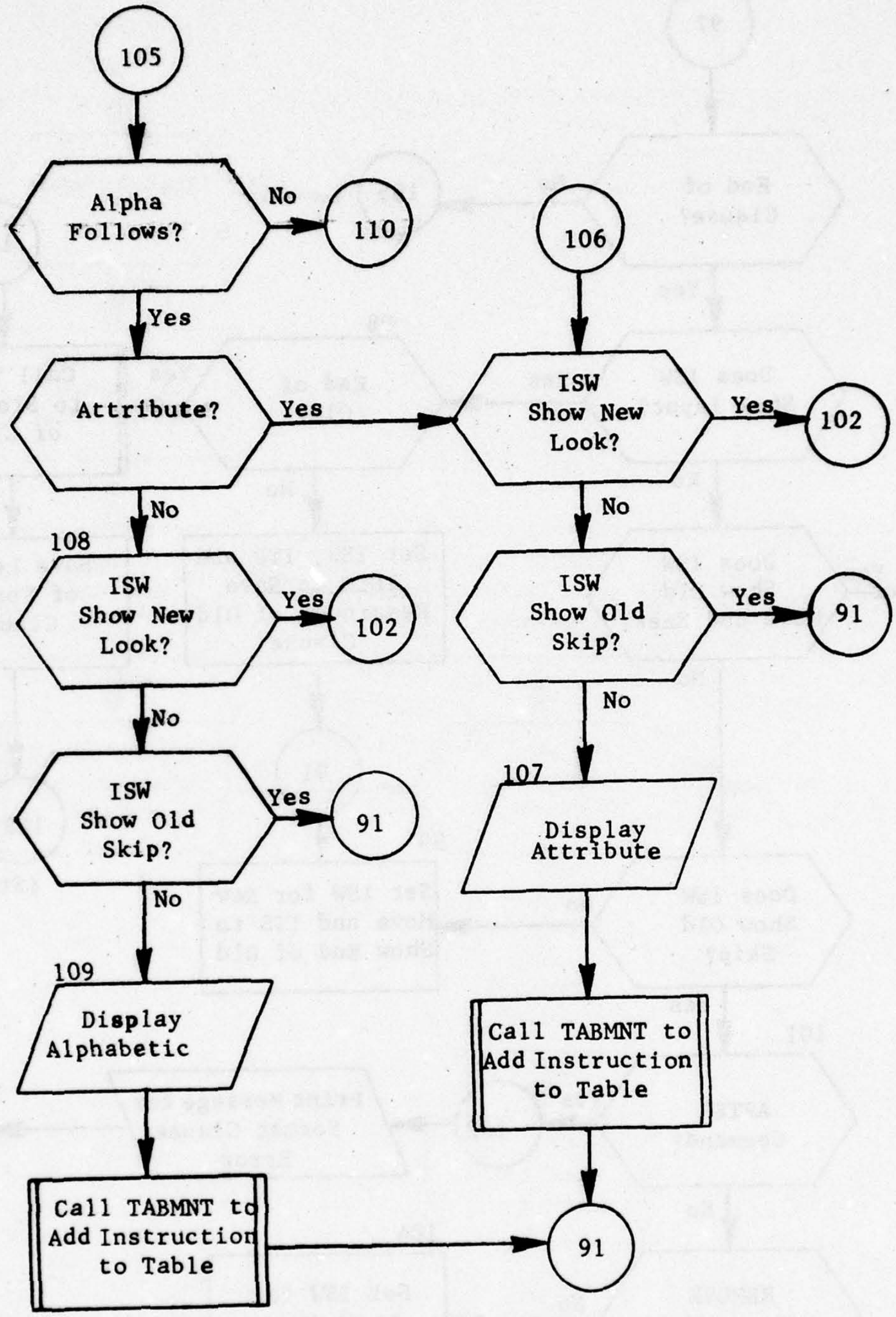


Figure 91. (Part 3 of 11)

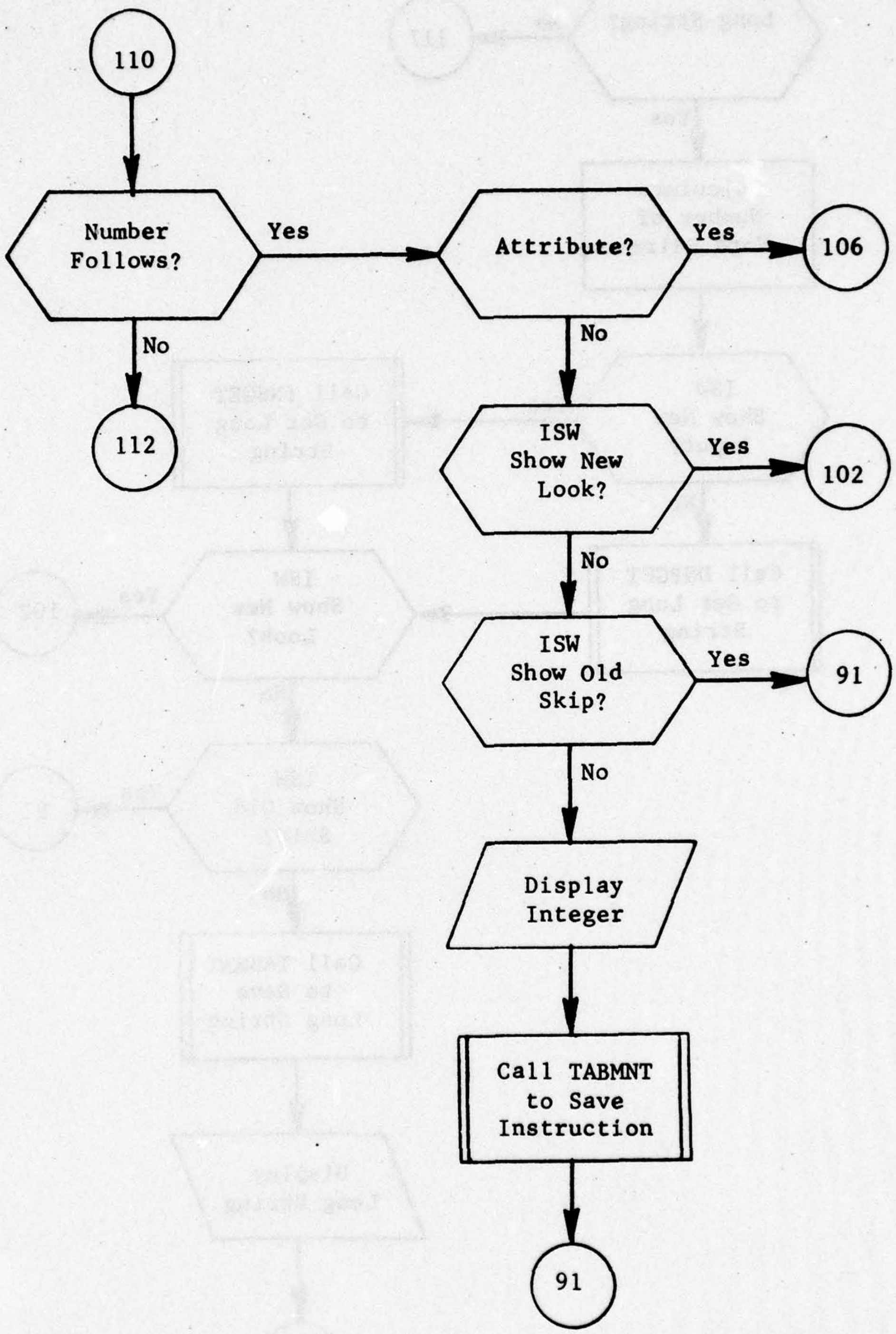


Figure 91. (Part 4 of 11)

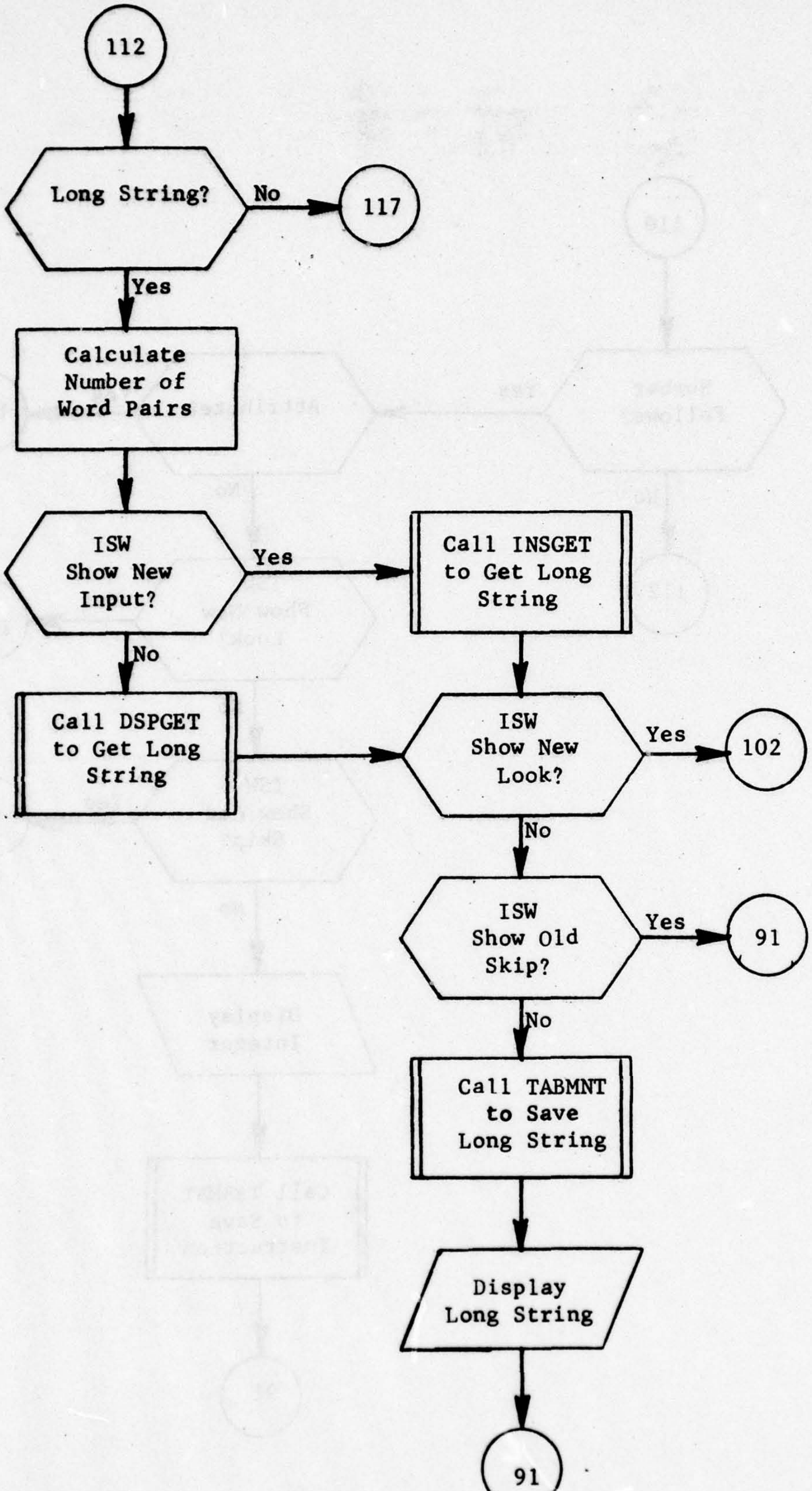


Figure 91. (Part 5 of 11)

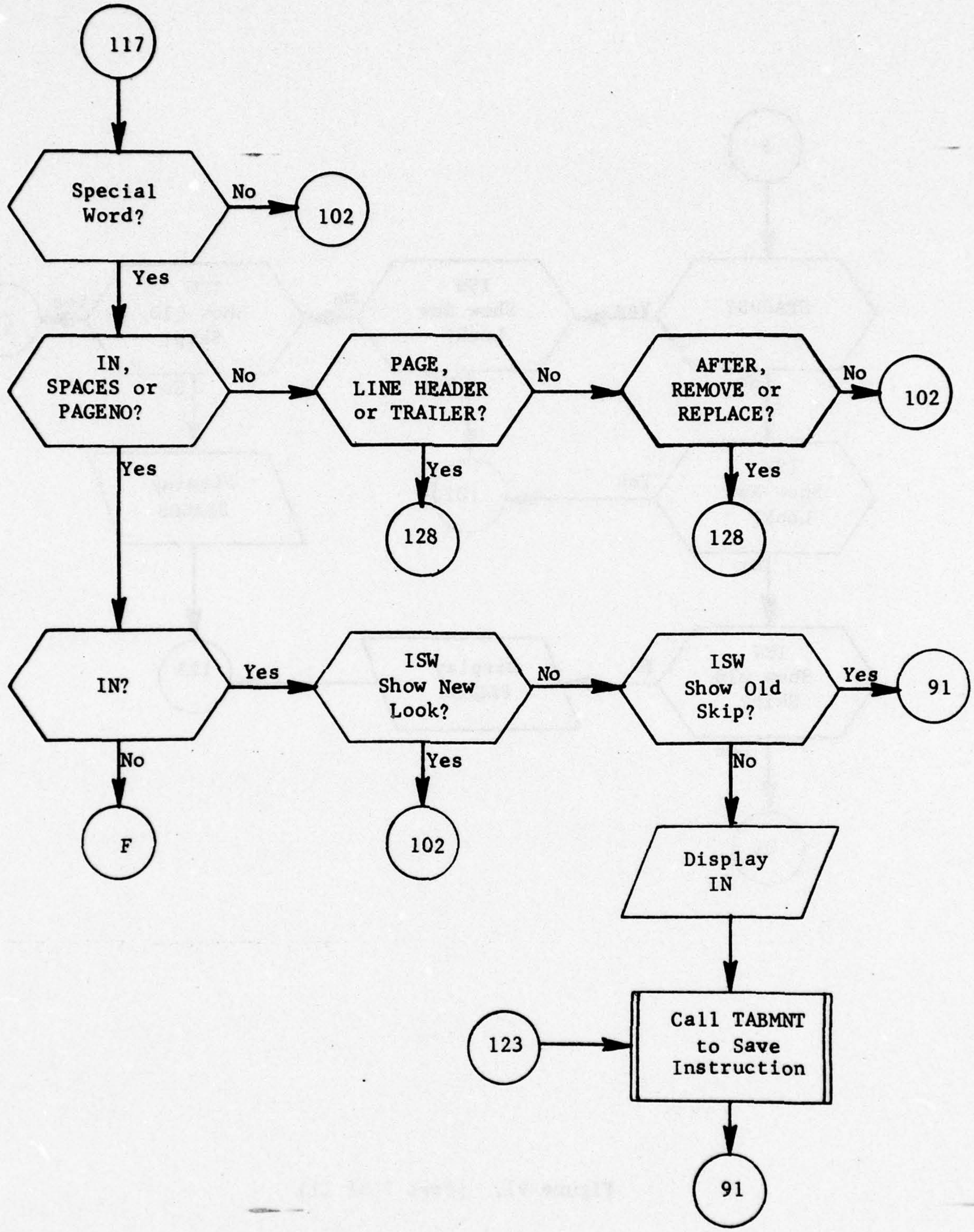


Figure 91. (Part 6 of 11)

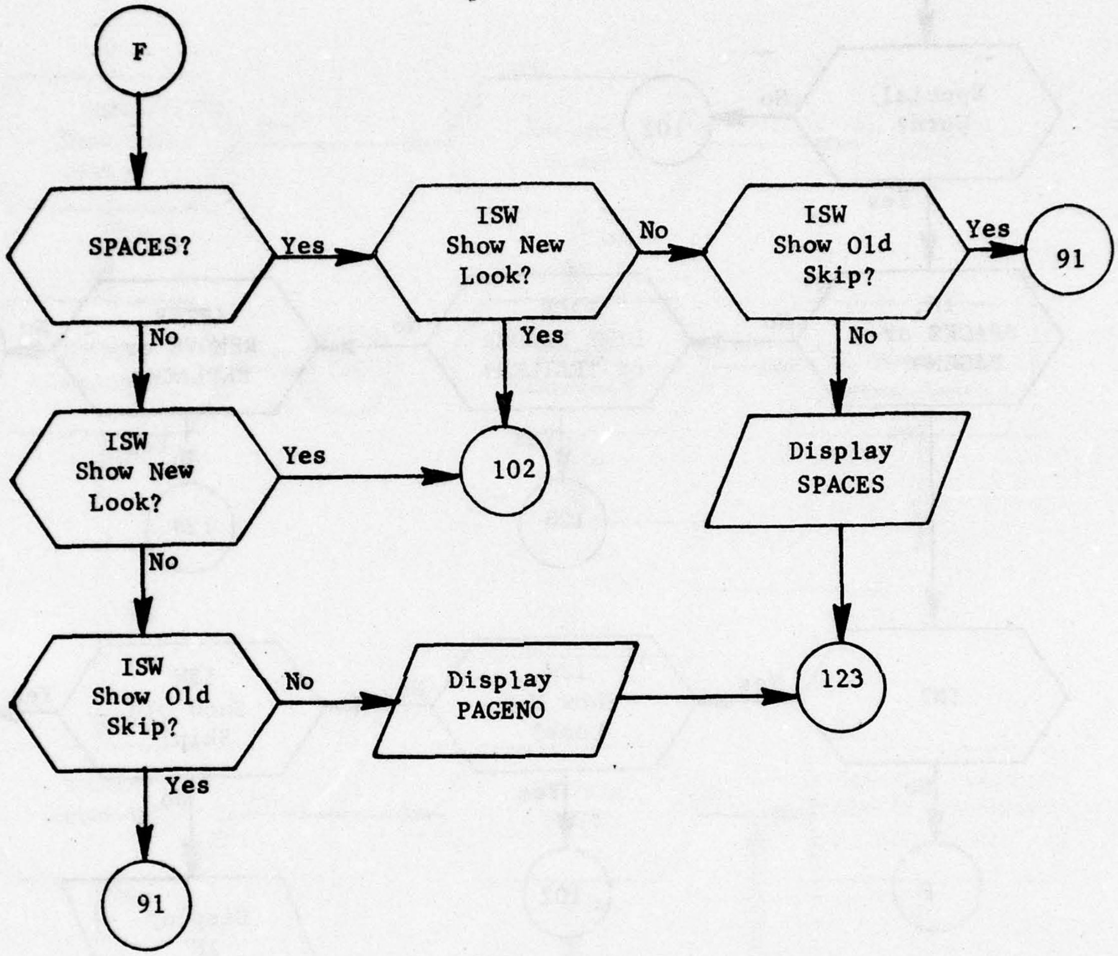


Figure 91. (Part 7 of 11)

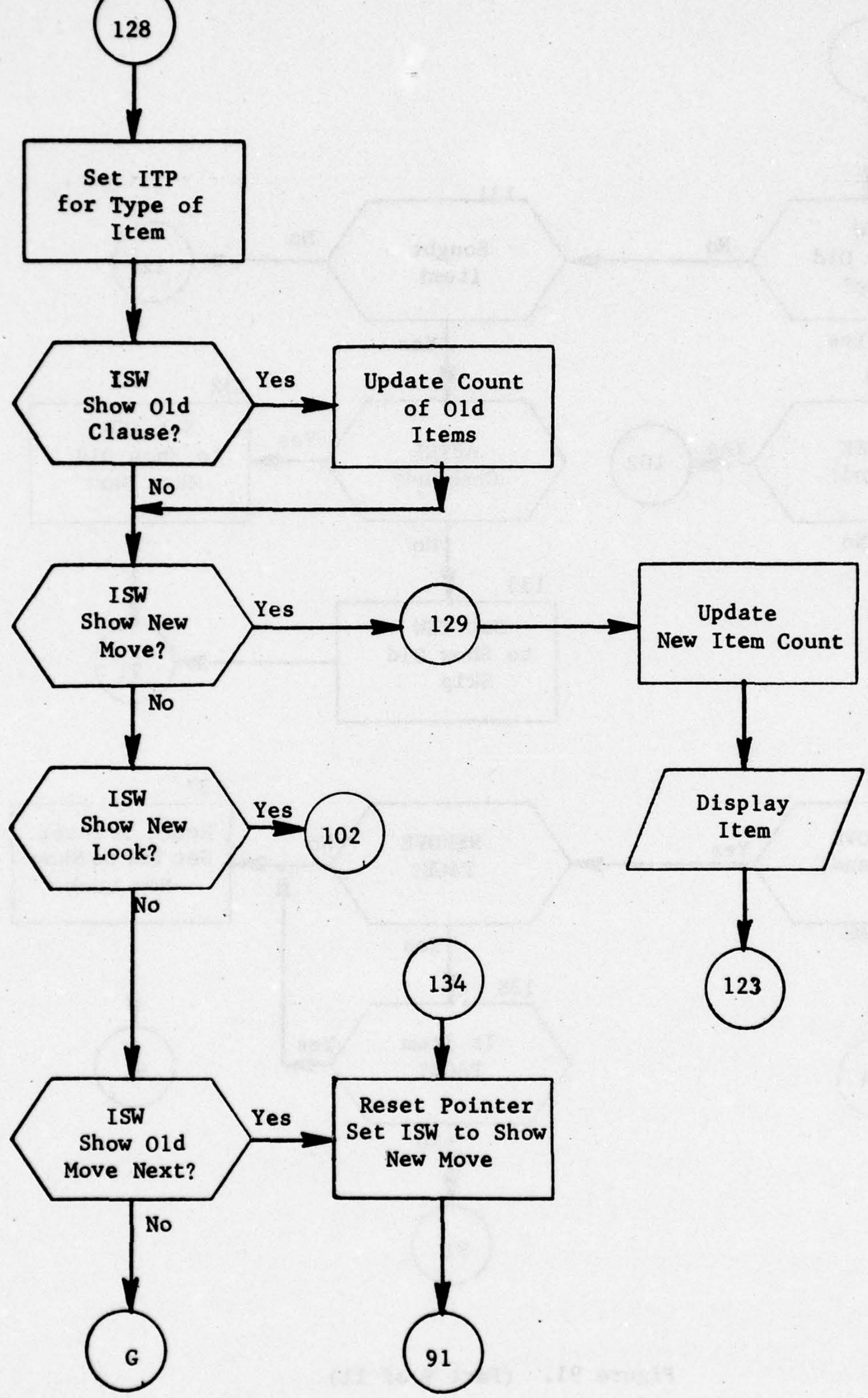


Figure 91. (Part 8 of 11)

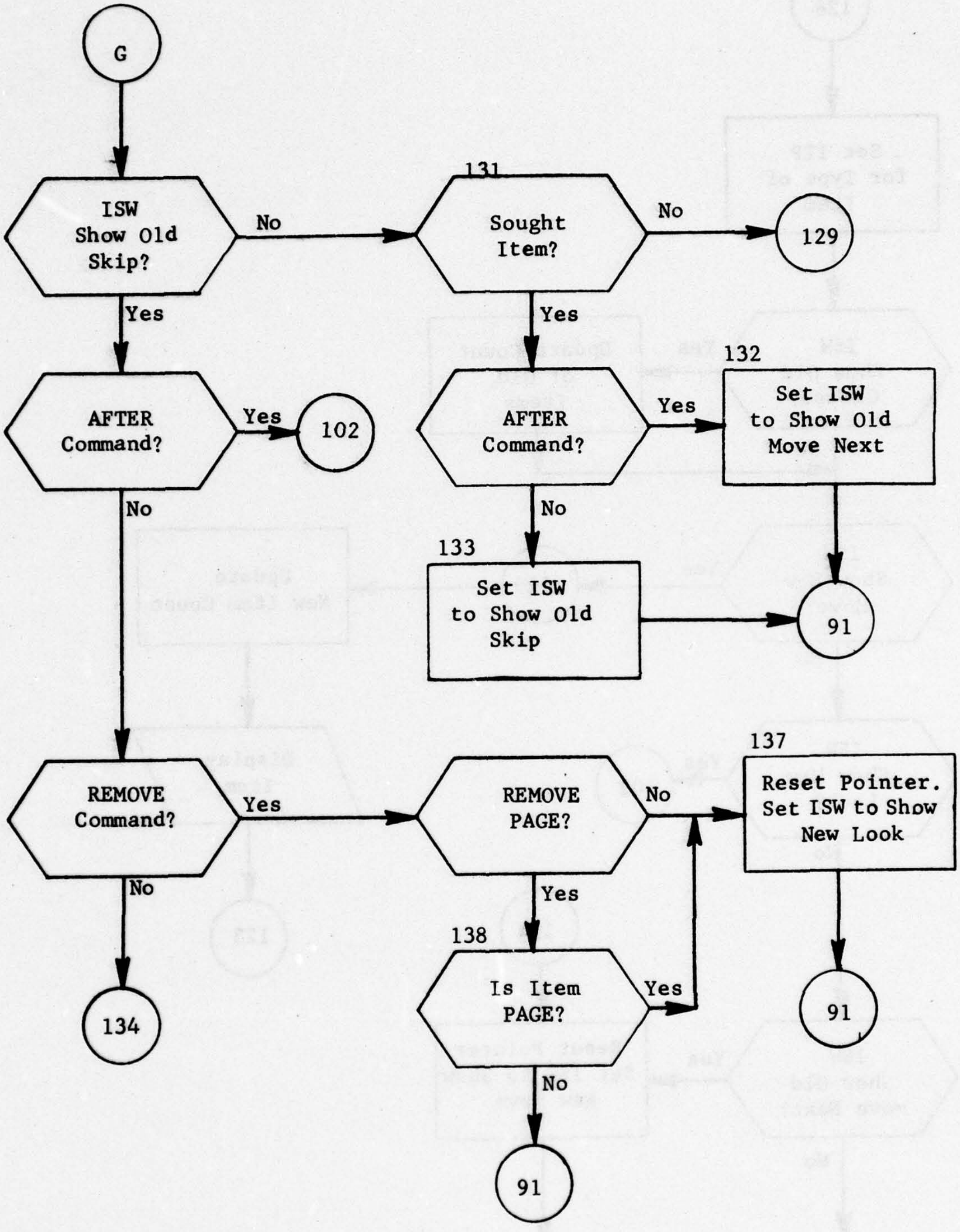


Figure 91. (Part 9 of 11)

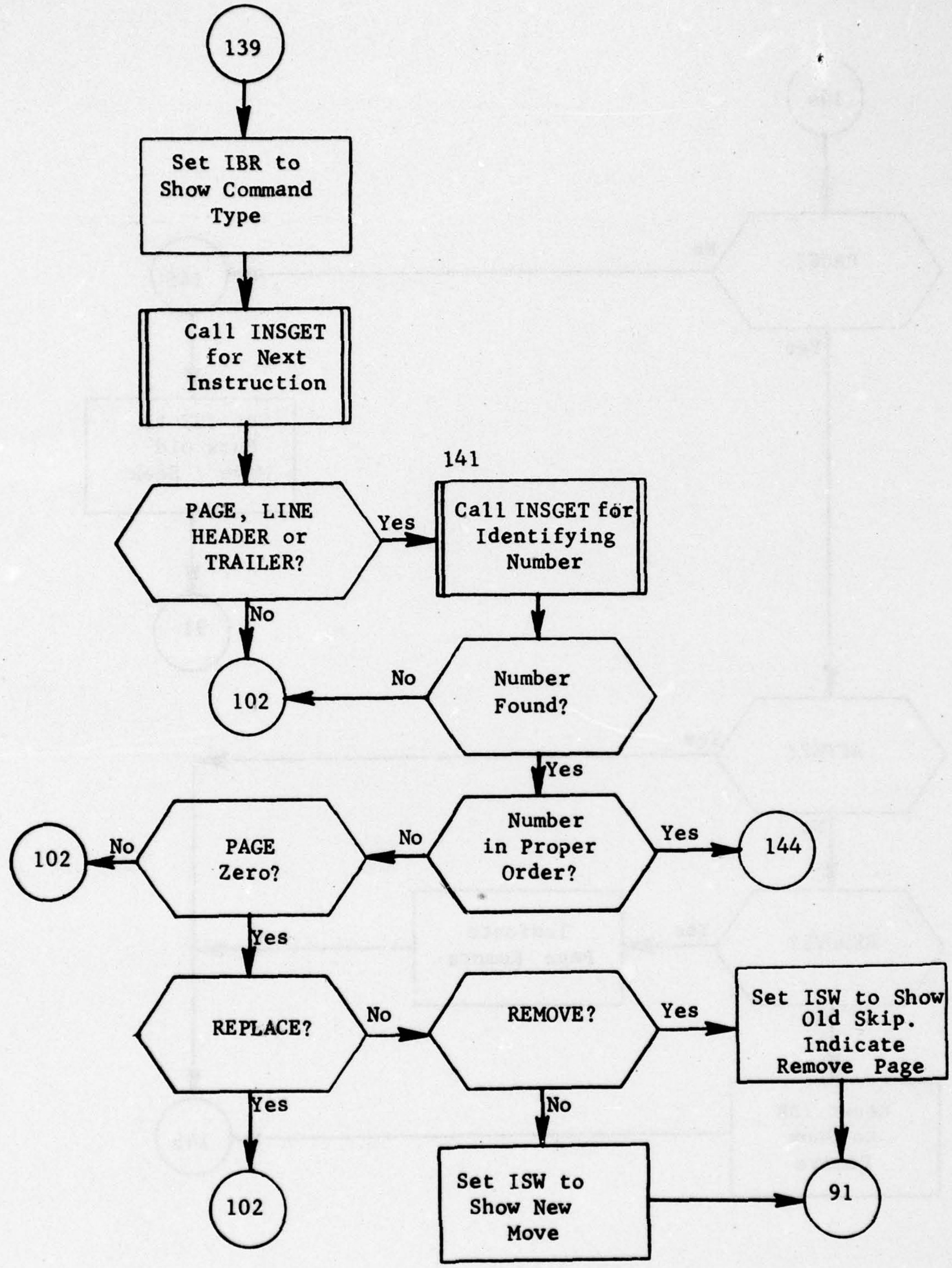


Figure 91. (Part 10 of 11)

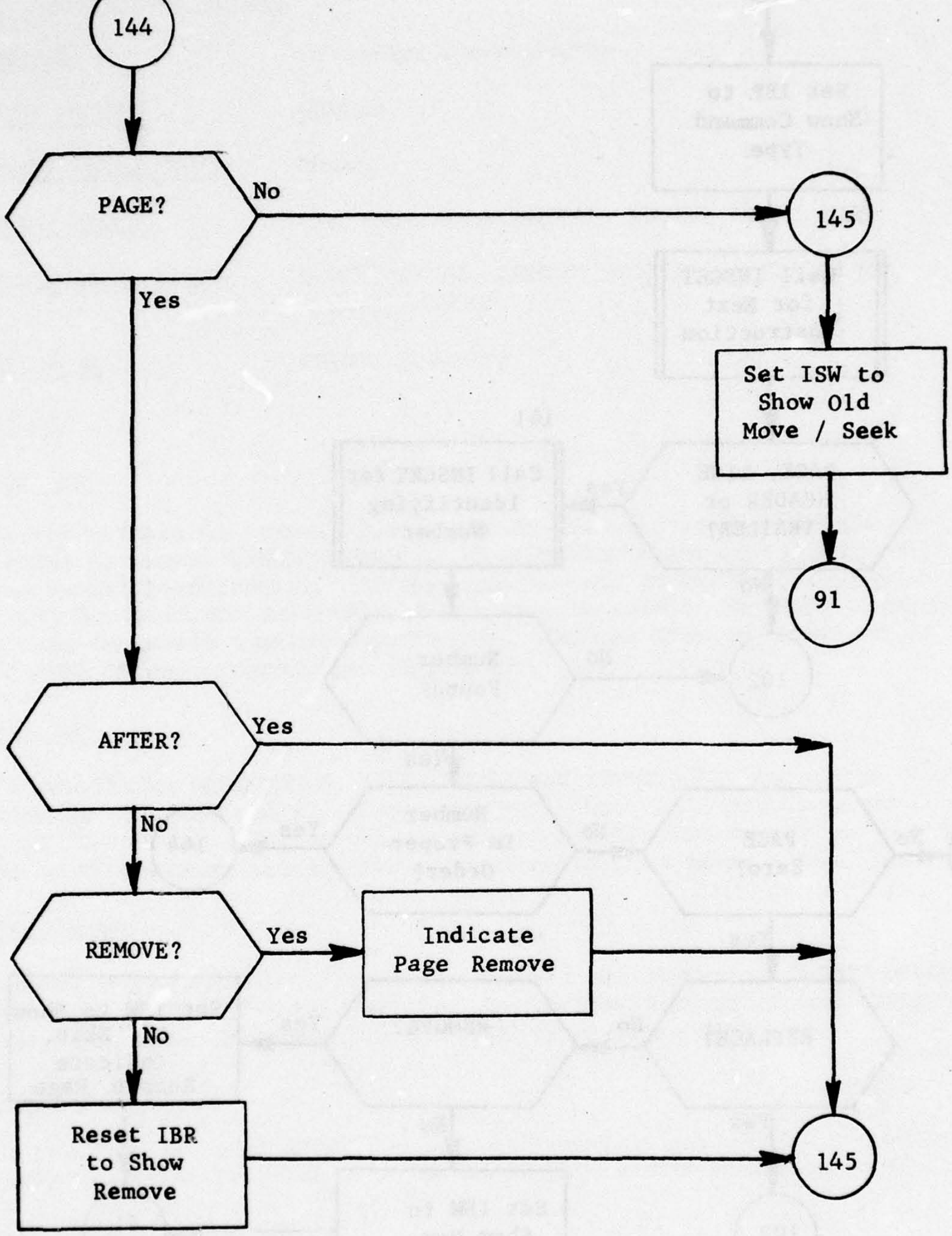


Figure 91. (Part 11 of 11)

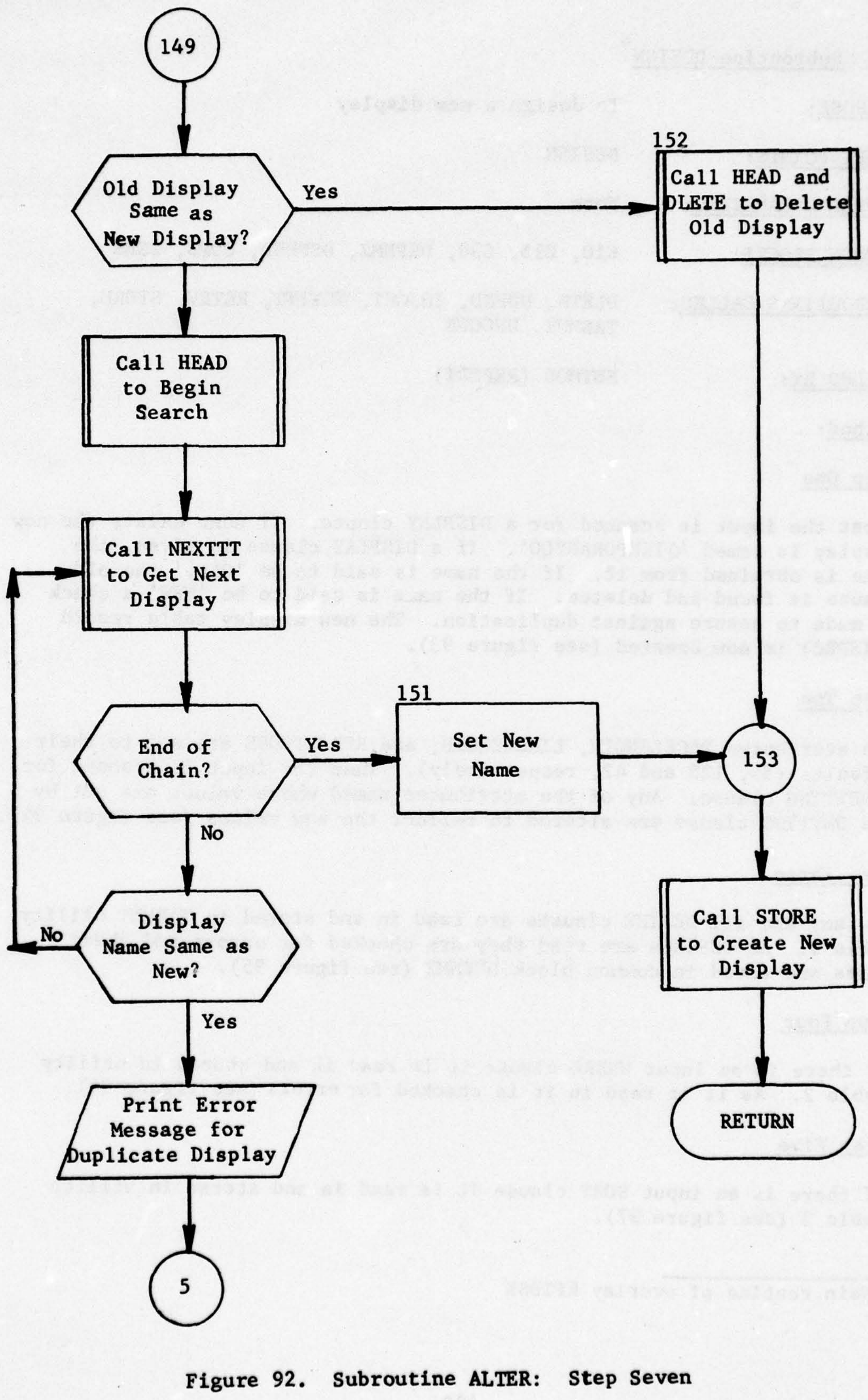


Figure 92. Subroutine ALTER: Step Seven

6.9 Subroutine DESIGN*

PURPOSE: To design a new display

ENTRY POINTS: DESIGN

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C15, C30, DEFNMZ, DSPHED, OOPS, ZEES

SUBROUTINES CALLED: DLETE, HDFND, INSGET, NEXTTT, RETRV, STORE, TABMNT, UNCODE

CALLED BY: ENTMOD (REPORT)

Method:

Step One

First the input is scanned for a DISPLAY clause. If none exists the new display is named 'QTEMPORARYQQ'. If a DISPLAY clause is given, the name is obtained from it. If the name is said to be 'OLD,' the old clause is found and deleted. If the name is said to be 'NEW' a check is made to assure against duplication. The new display table record (DISPRC) is now created (see figure 93).

Step Two

The attributes PAGELENGTH, LINELENGTH, and REPORTCODE are set to their defaults (55, 120 and 42, respectively). Then the input is scanned for a SETTING clause. Any of the attributes named whose values are set by the SETTING clause are altered to reflect the new values (see figure 94).

Step Three

Now any and all DEFINE clauses are read in and stored in TABMNT utility table 1. As DEFINEs are read they are checked for errors and their names are saved in common block DEFNMZ (see figure 95).

Step Four

If there is an input WHERE clause it is read in and stored in utility table 2. As it is read in it is checked for errors (see figure 96).

Step Five

If there is an input SORT clause it is read in and stored in utility table 3 (see figure 97).

*Main routine of overlay RPTDSN

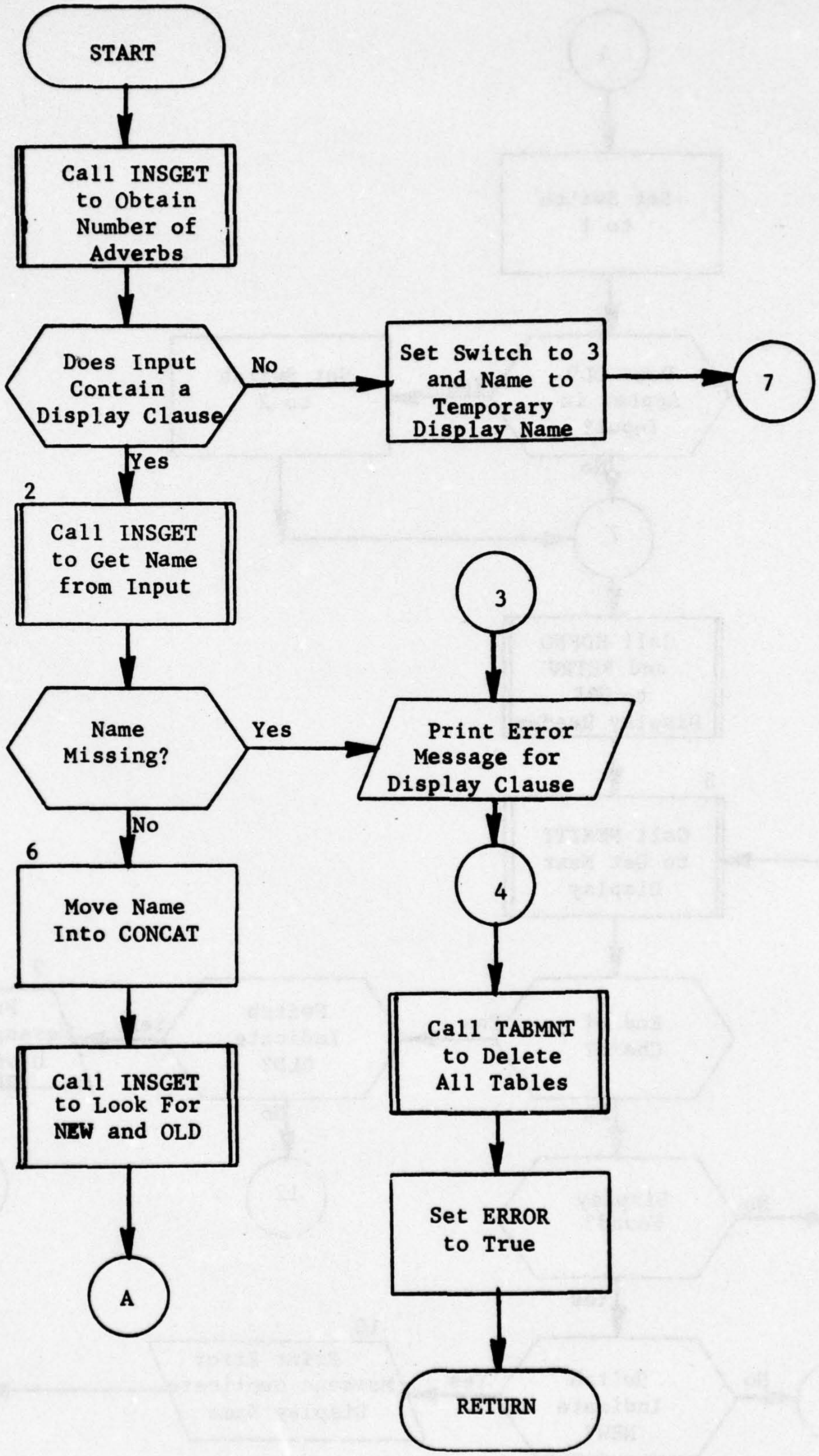


Figure 93. Subroutine DESIGN: Step One (Part 1 of 3)

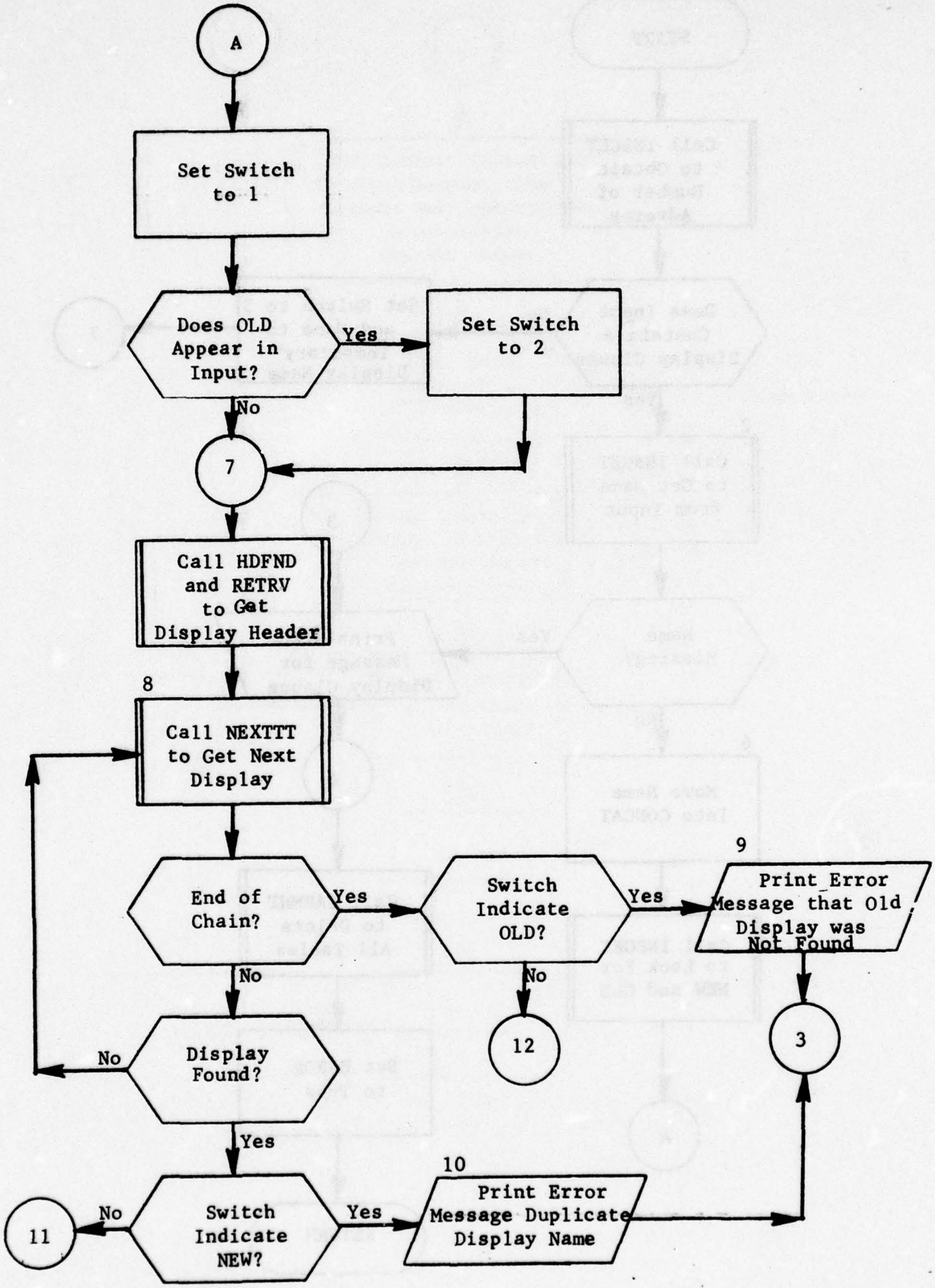


Figure 93. (Part 2 of 3)

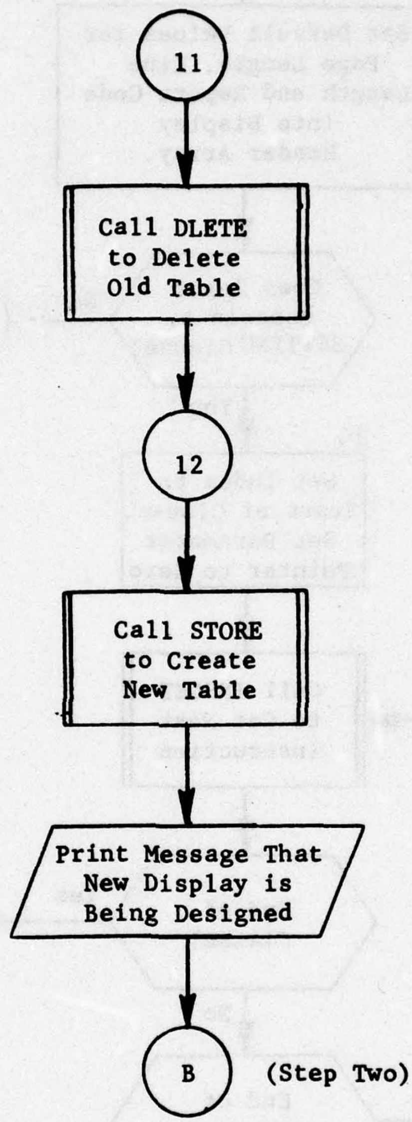


Figure 93. (Part 3 of 3)

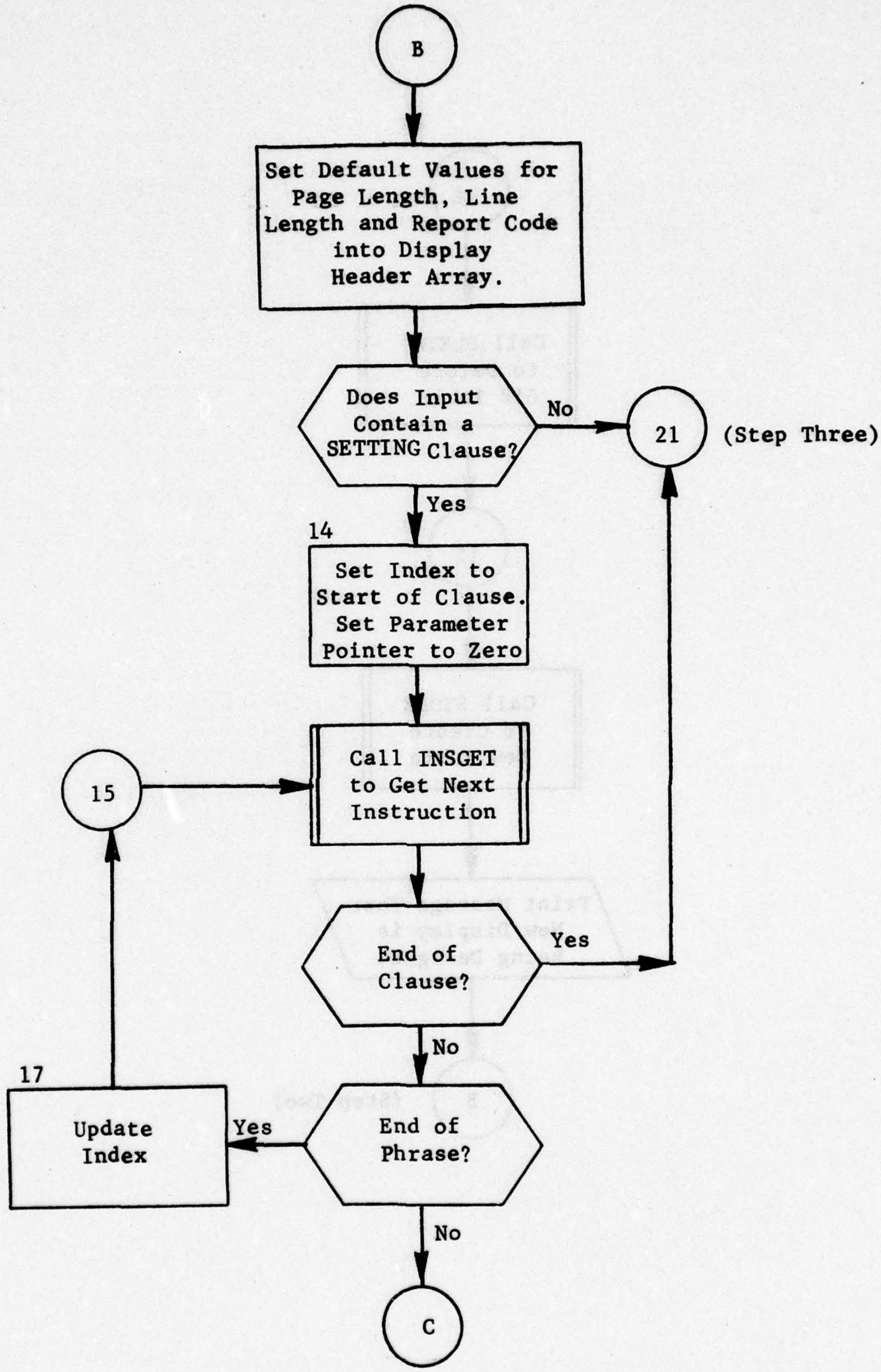


Figure 94. Subroutine DESIGN: Step Two (Part 1 of 2)

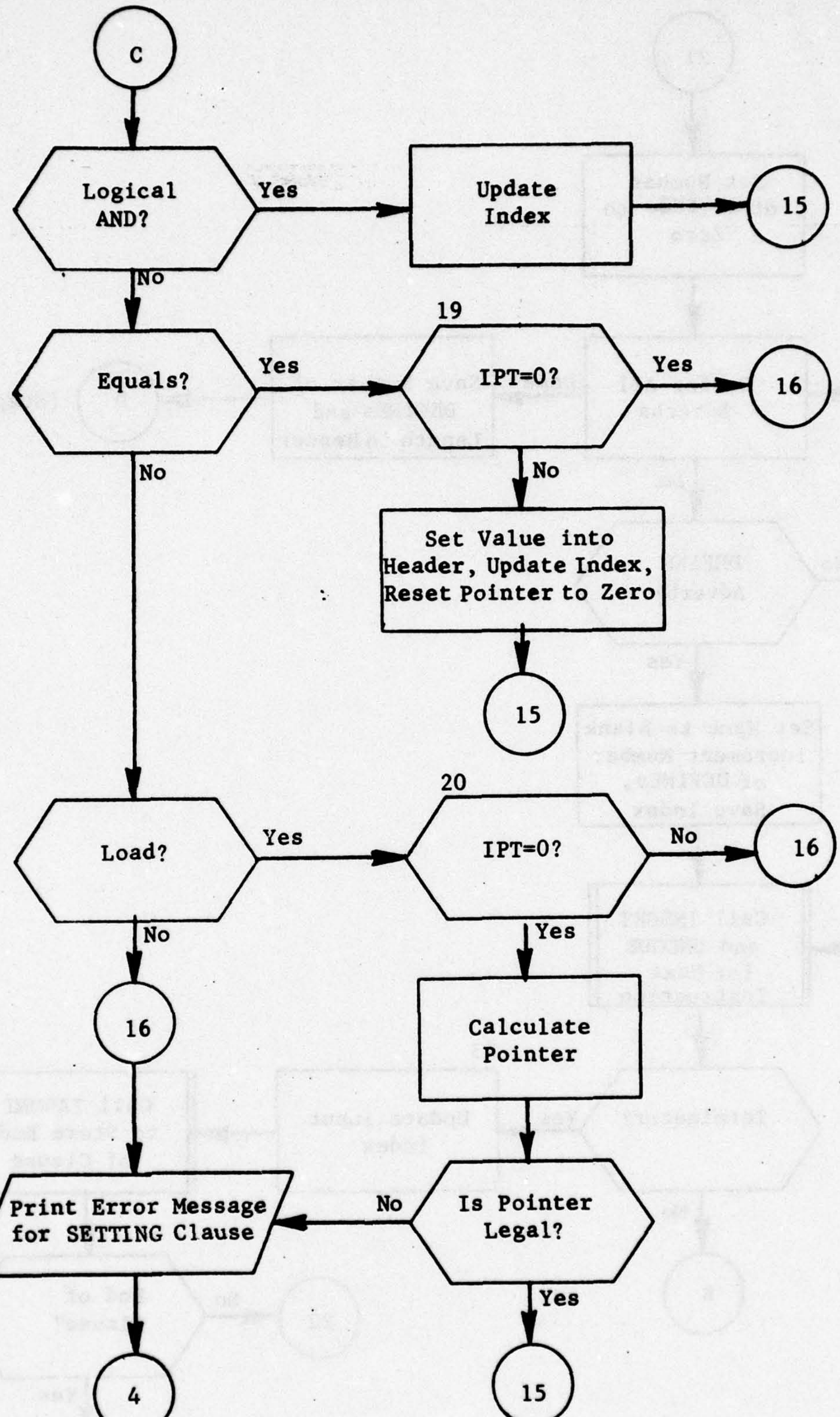


Figure 94. (Part 2 of 2)

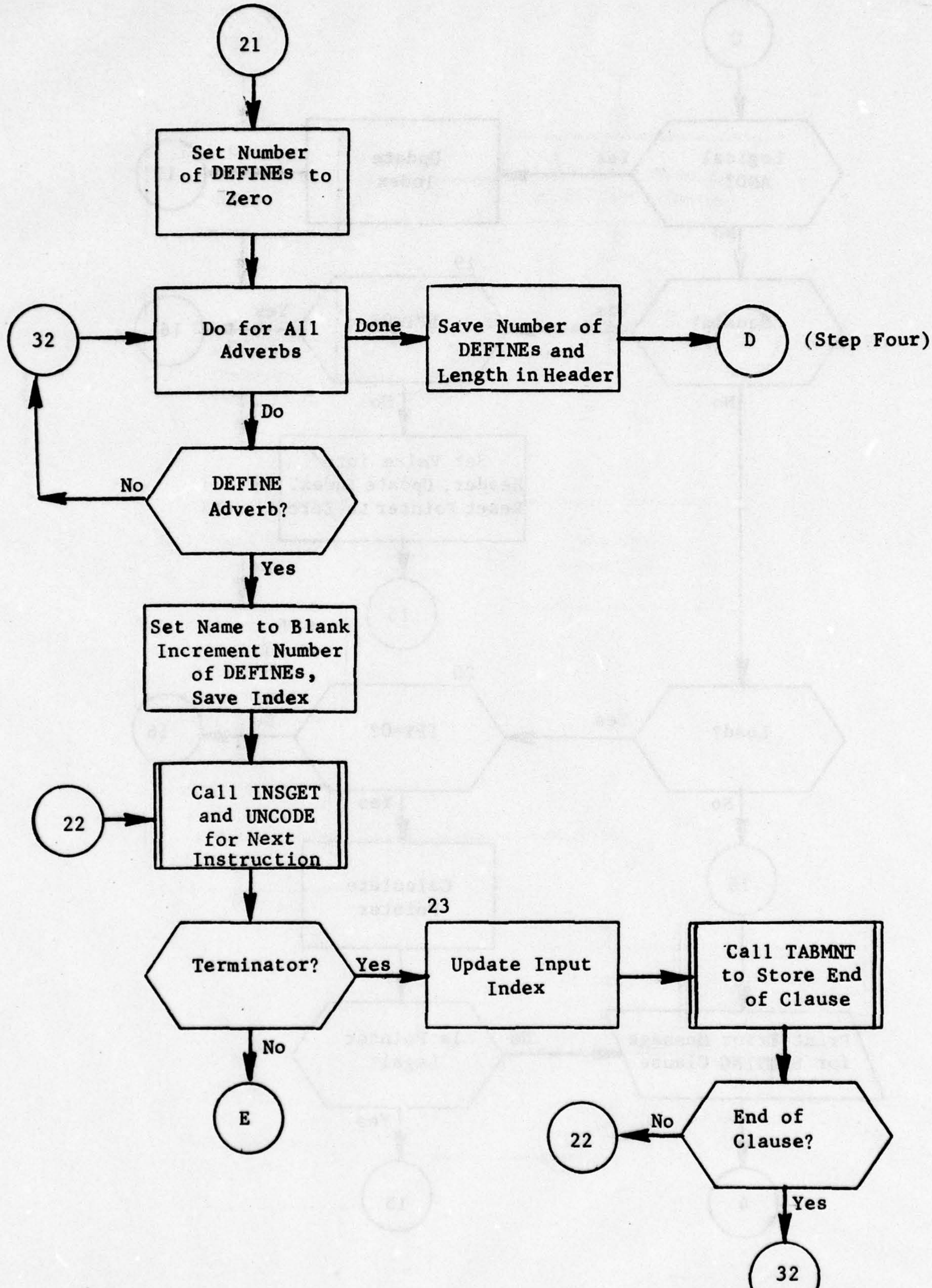


Figure 95. Subroutine DESIGN: Step Three (Part 1 of 3)

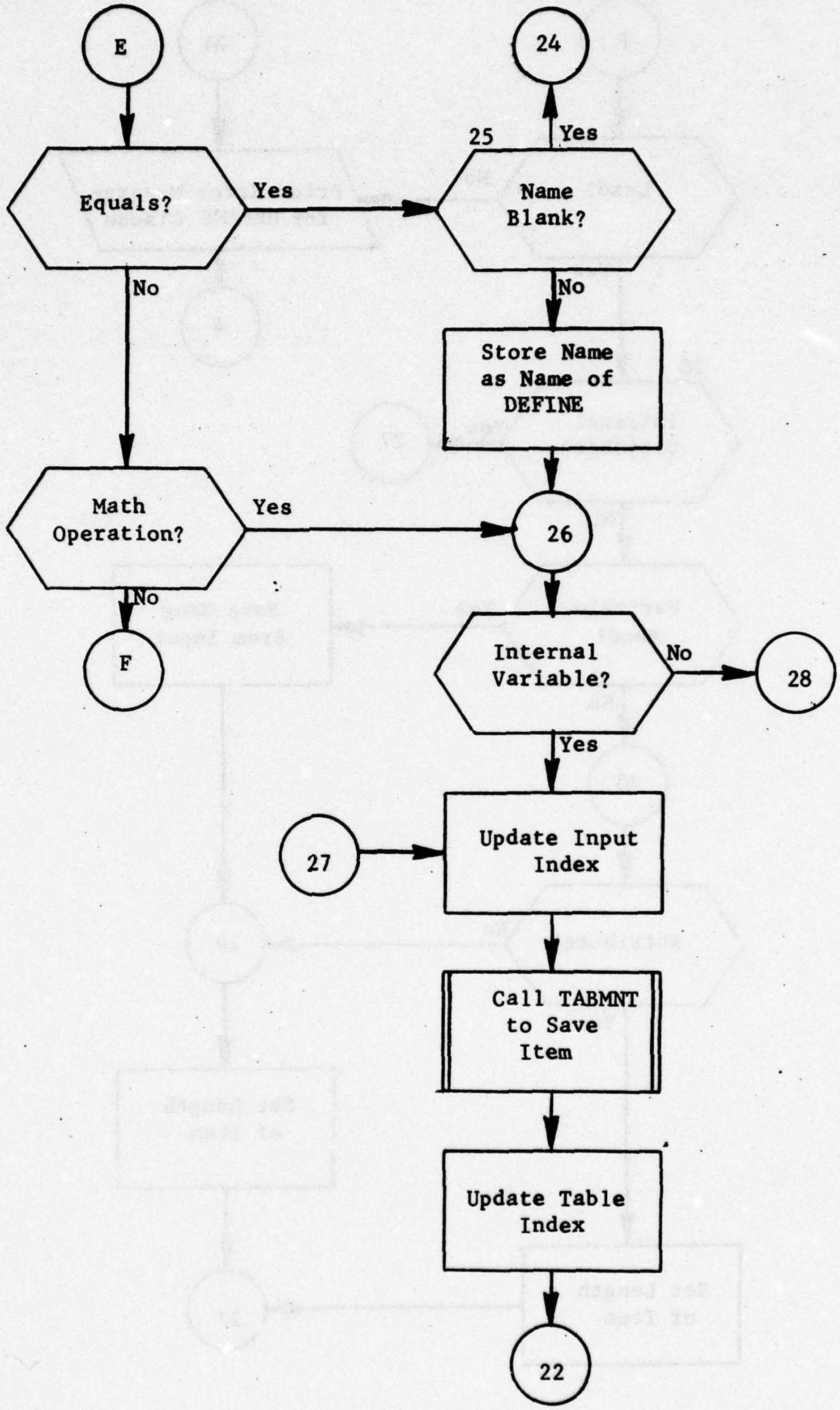


Figure 95. (Part 2 of 3)

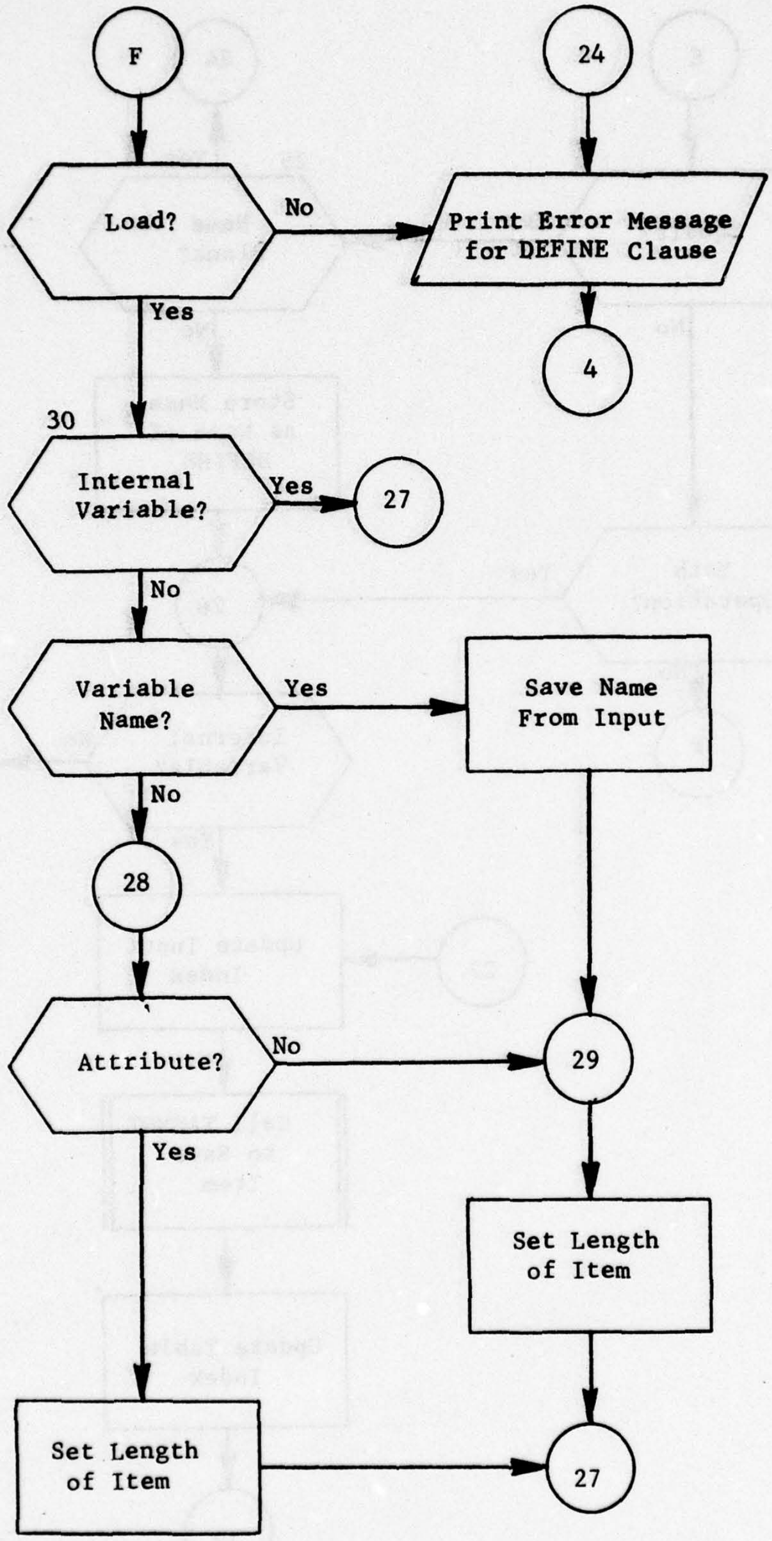


Figure 95. (Part 3 of 3)

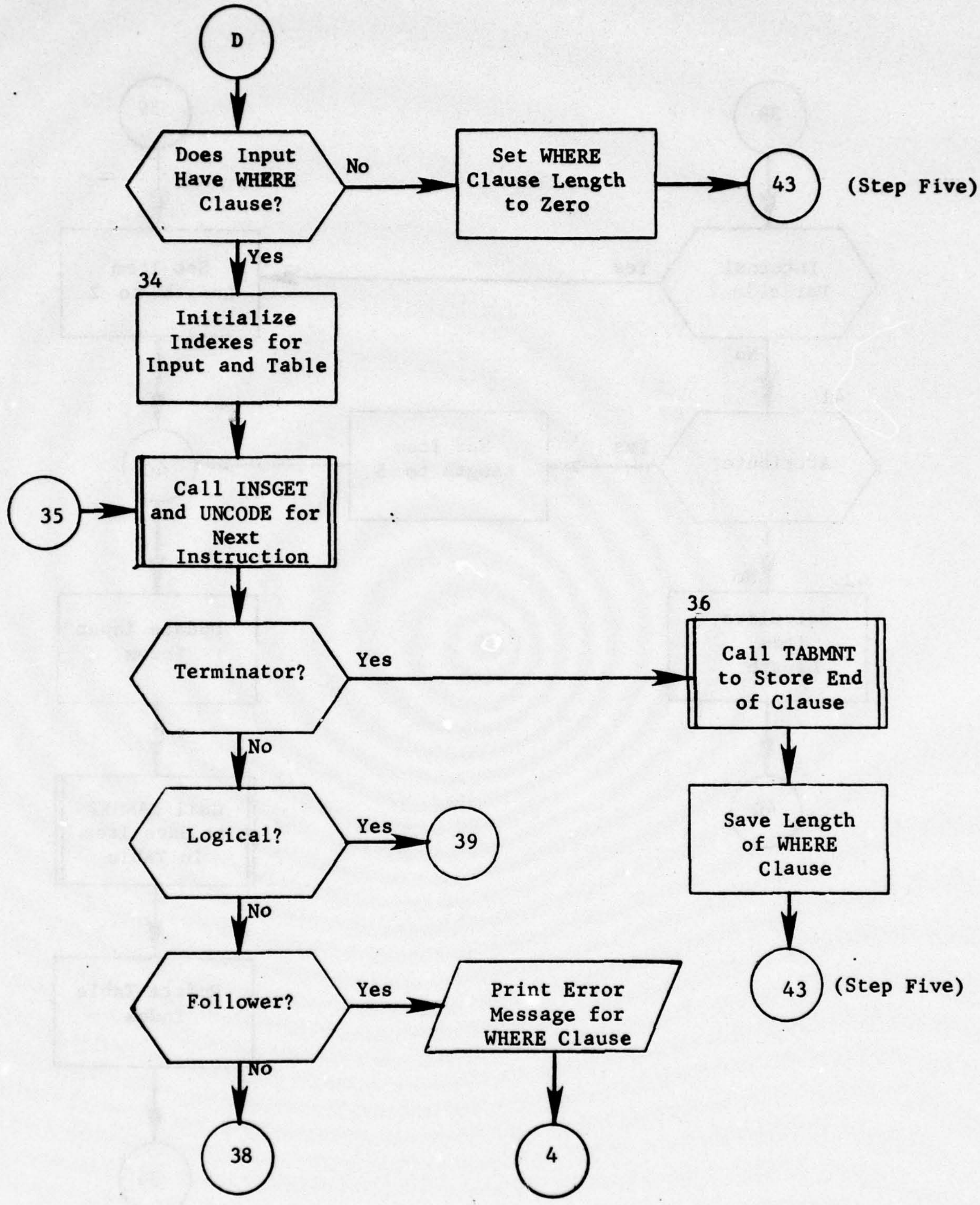


Figure 96. Subroutine DESIGN: Step Four (Part 1 of 2)

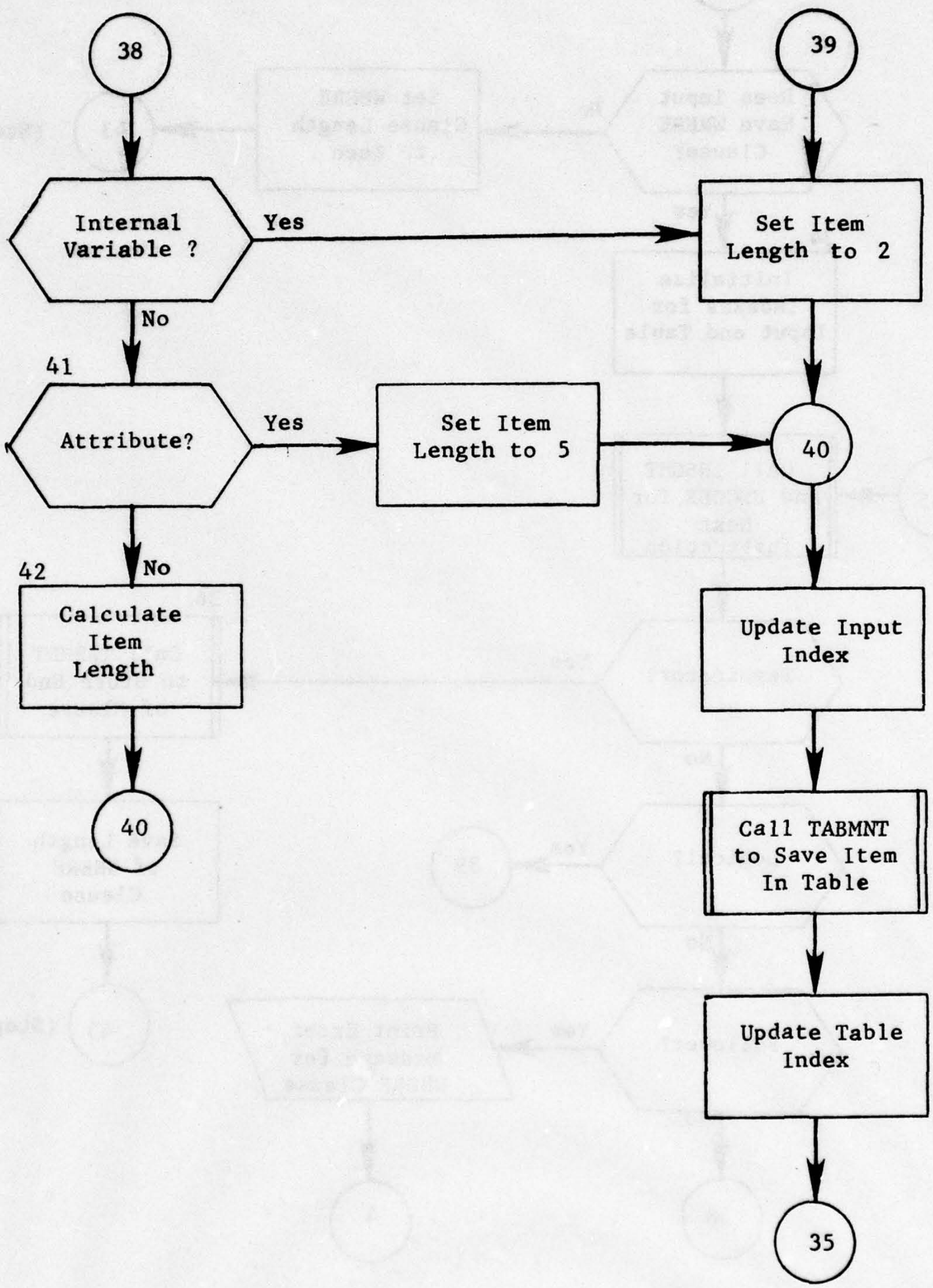


Figure 96. (Part 2 of 2)

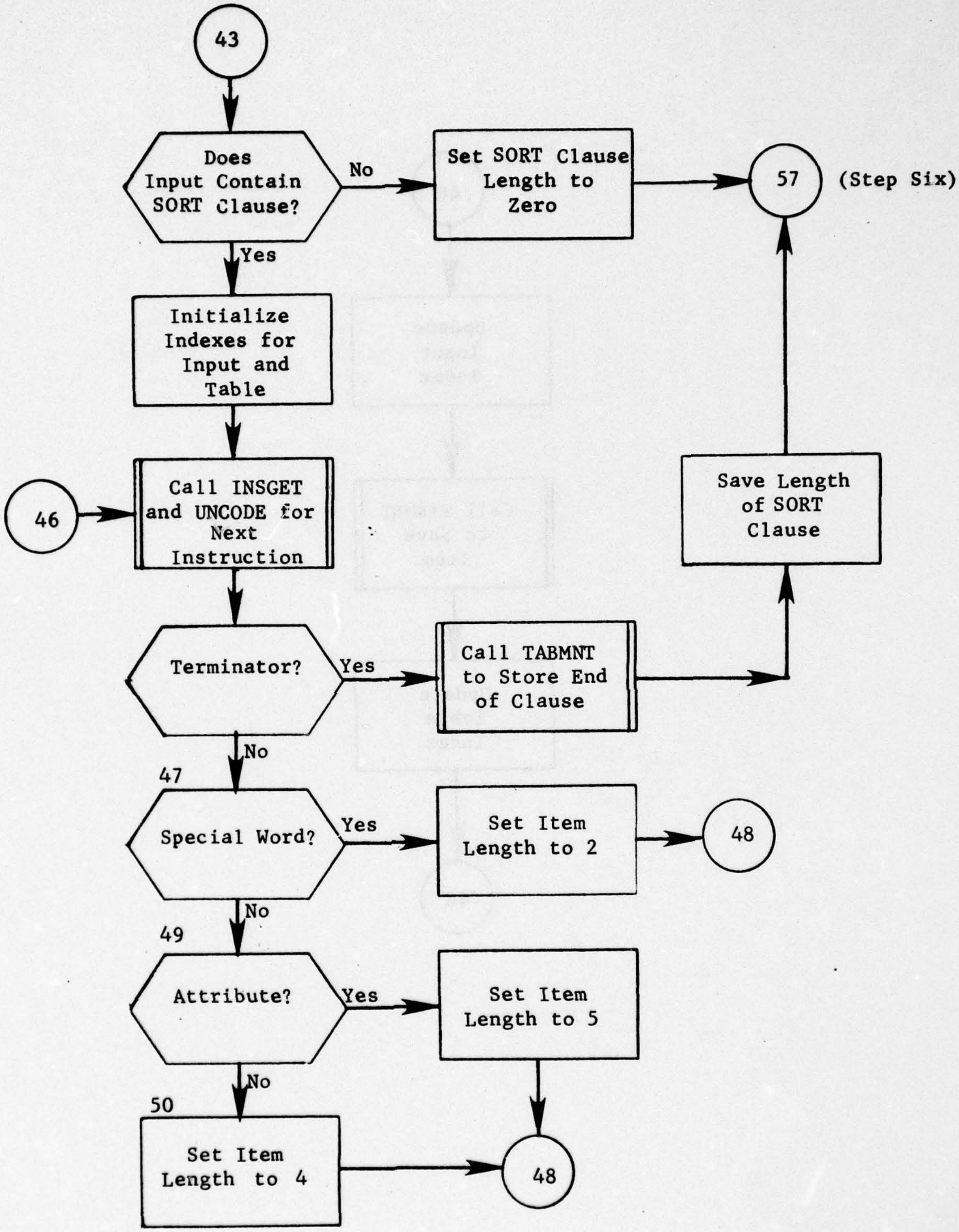


Figure 97. Subroutine DESIGN: Step Five (Part 1 of 2)


48
Update
Input
Index
Call TABMNT
to Save
Item
Update
Table
Index
46

Step Six

The FORMAT clause is now read in and stored in utility table 4. As it is read in, each item (PAGE, LINE, HEADER, TRAILER) is numbered and printed and each element (DEFINE name, attribute, long string, etc.) is checked for errors and printed (see figure 98).

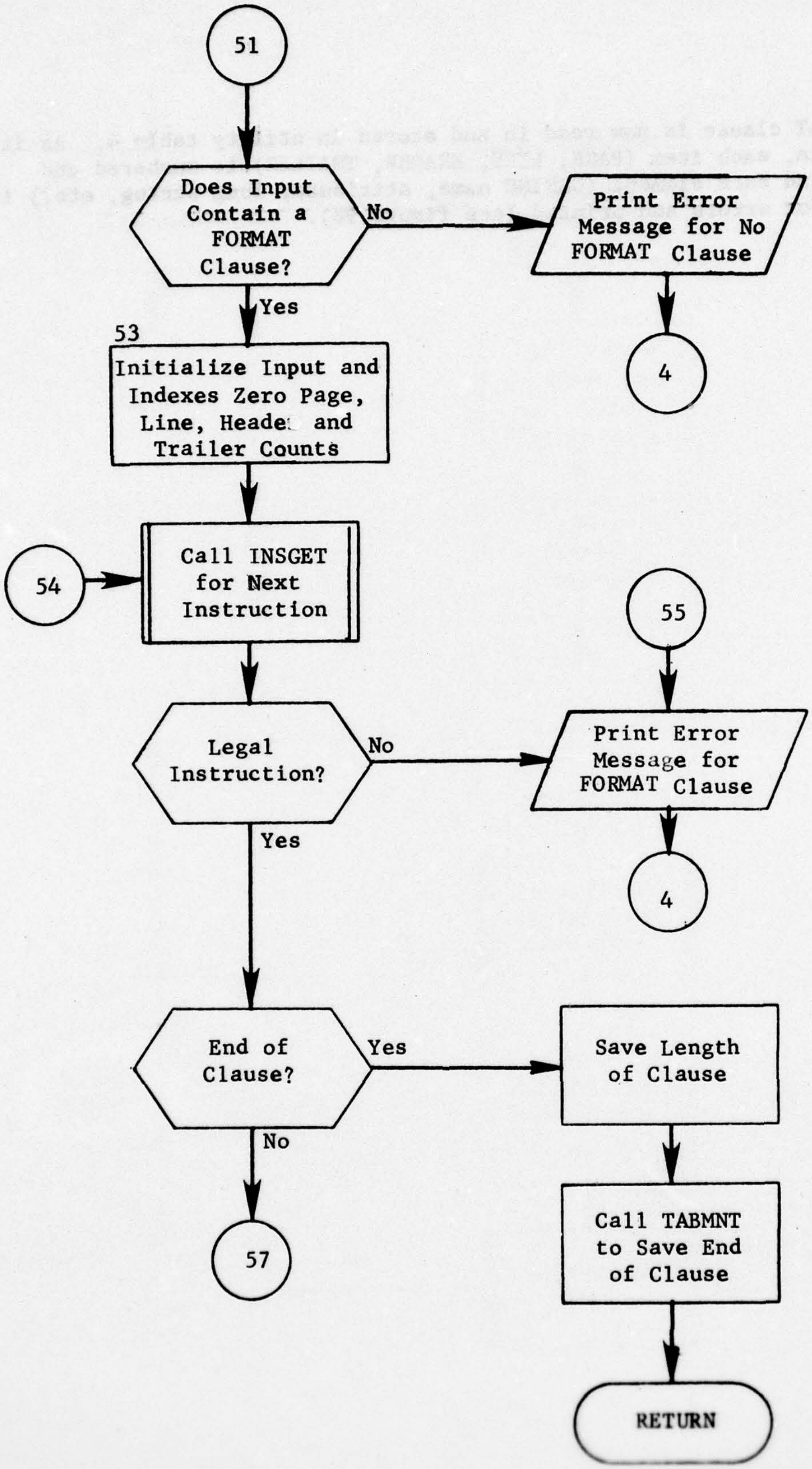


Figure 98. Subroutine DESIGN: Step Six (Part 1 of 5)

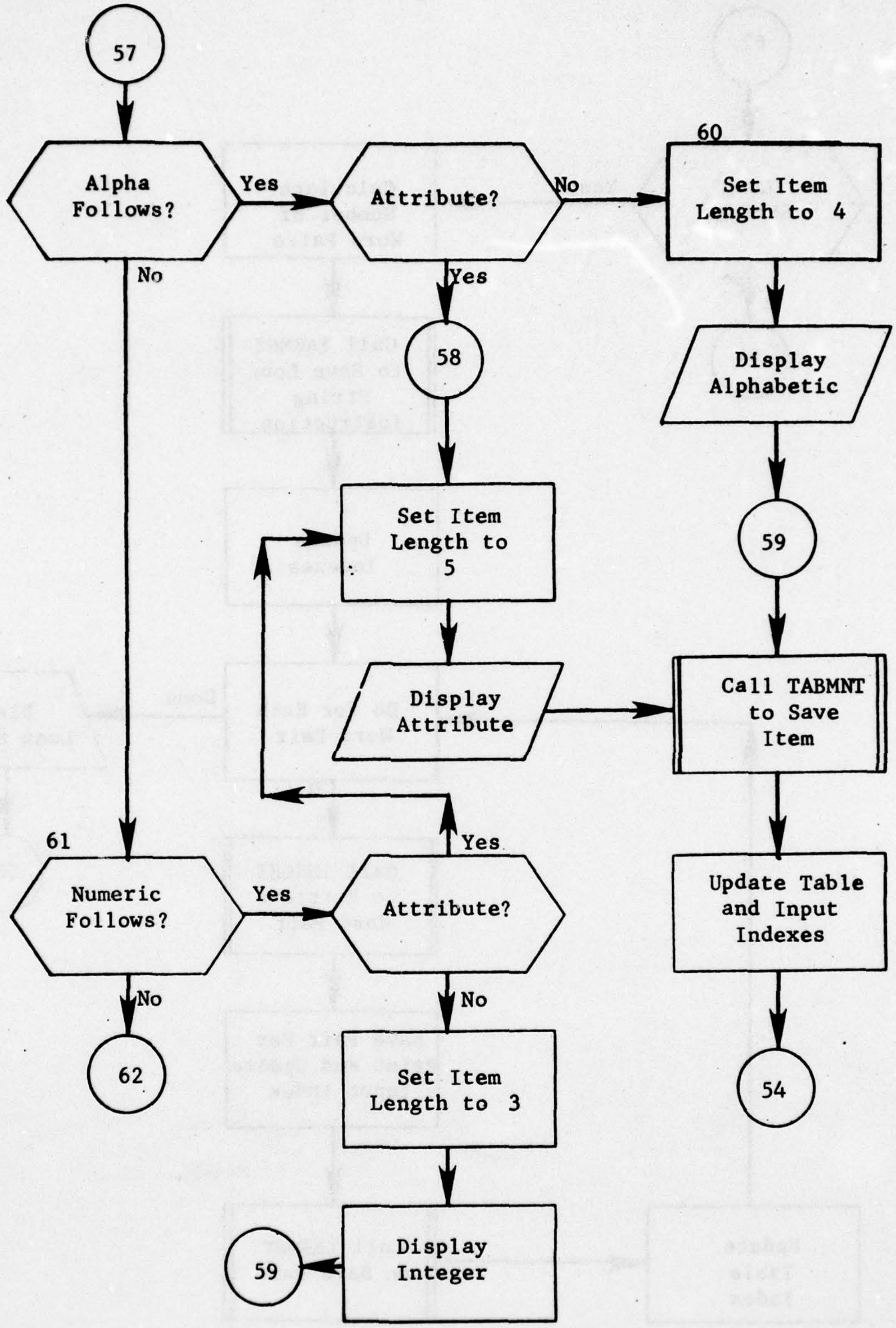


Figure 98. (Part 2 of 5)

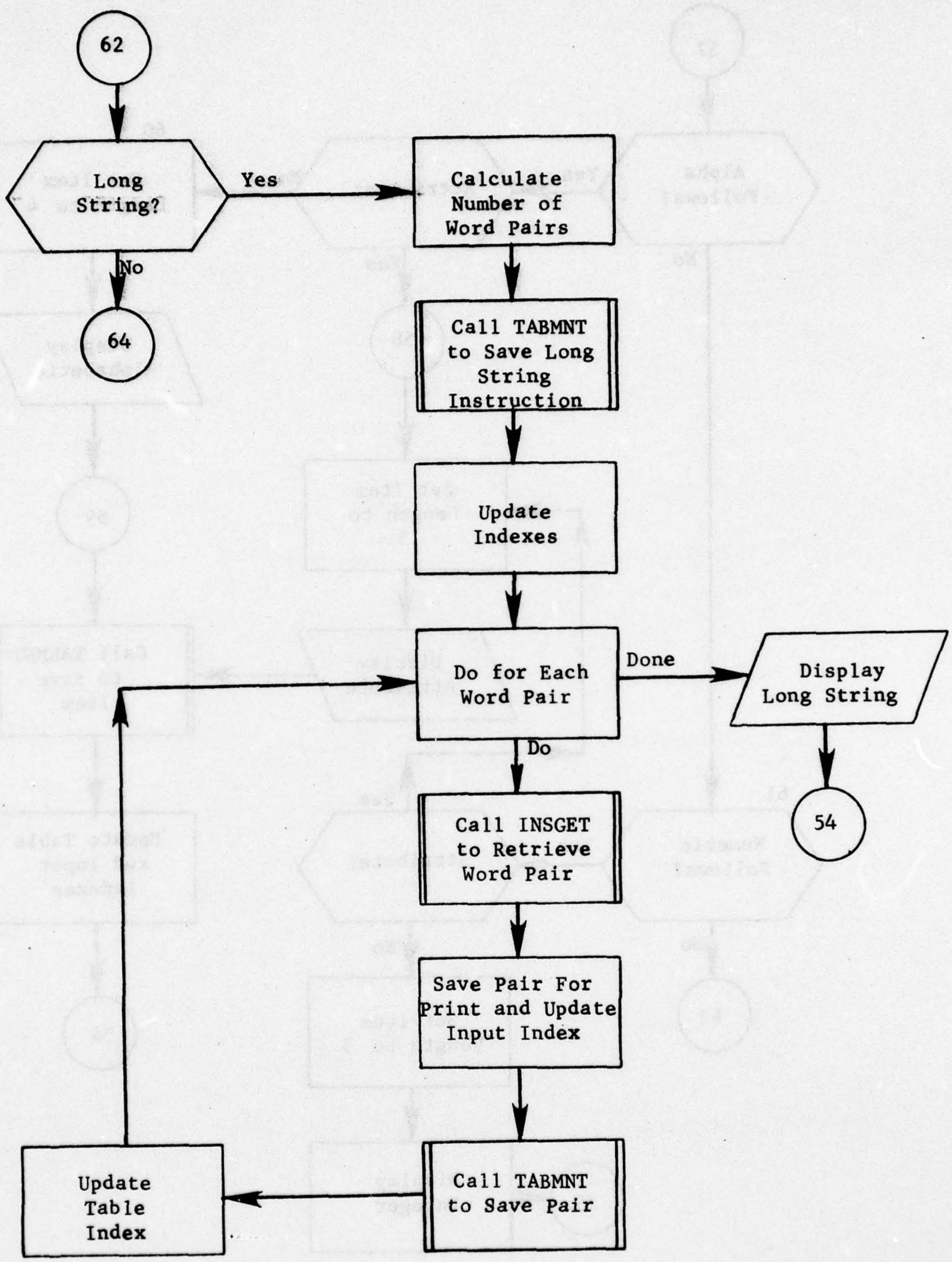


Figure 98. (Part 3 of 5)

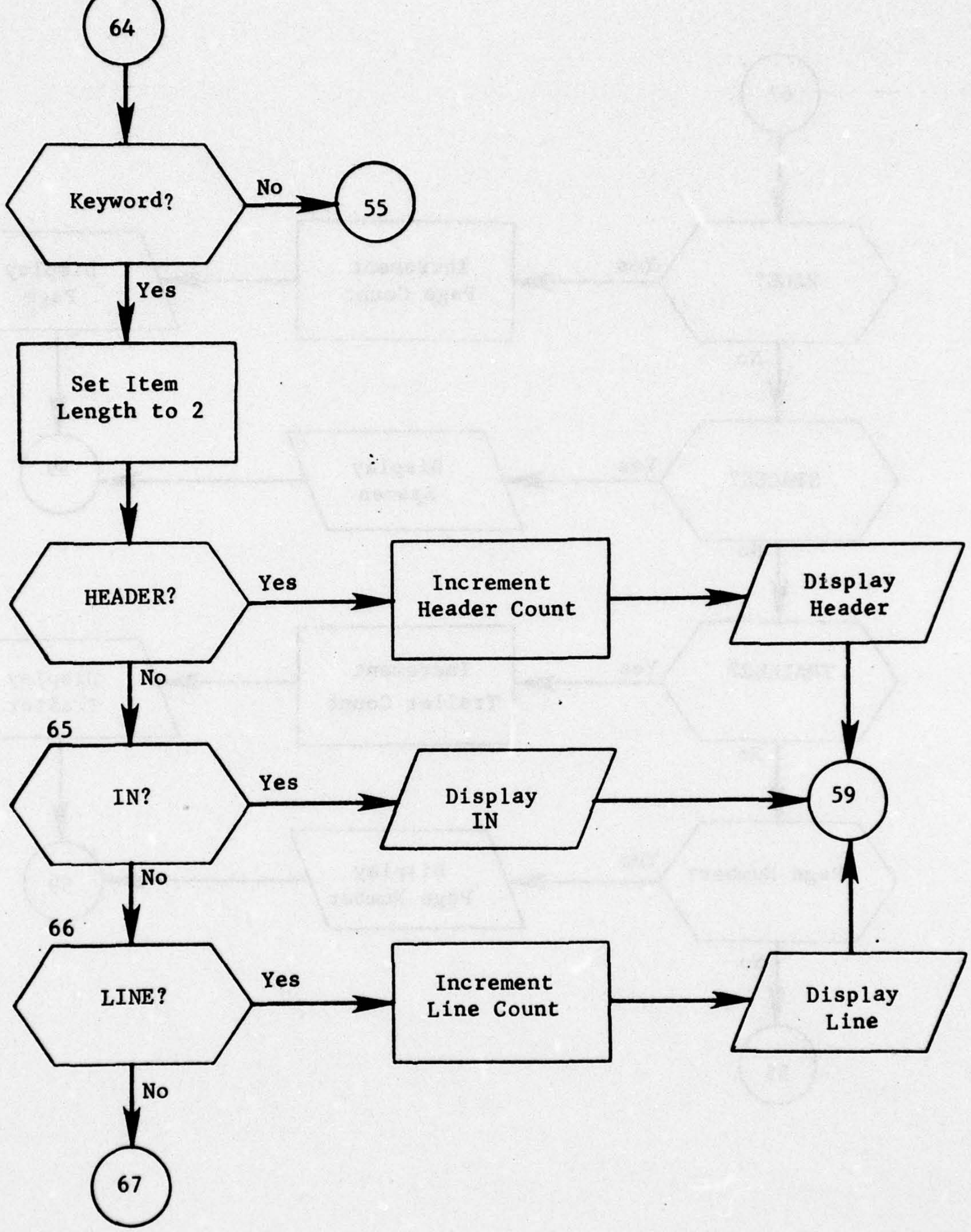


Figure 98. (Part 4 of 5)

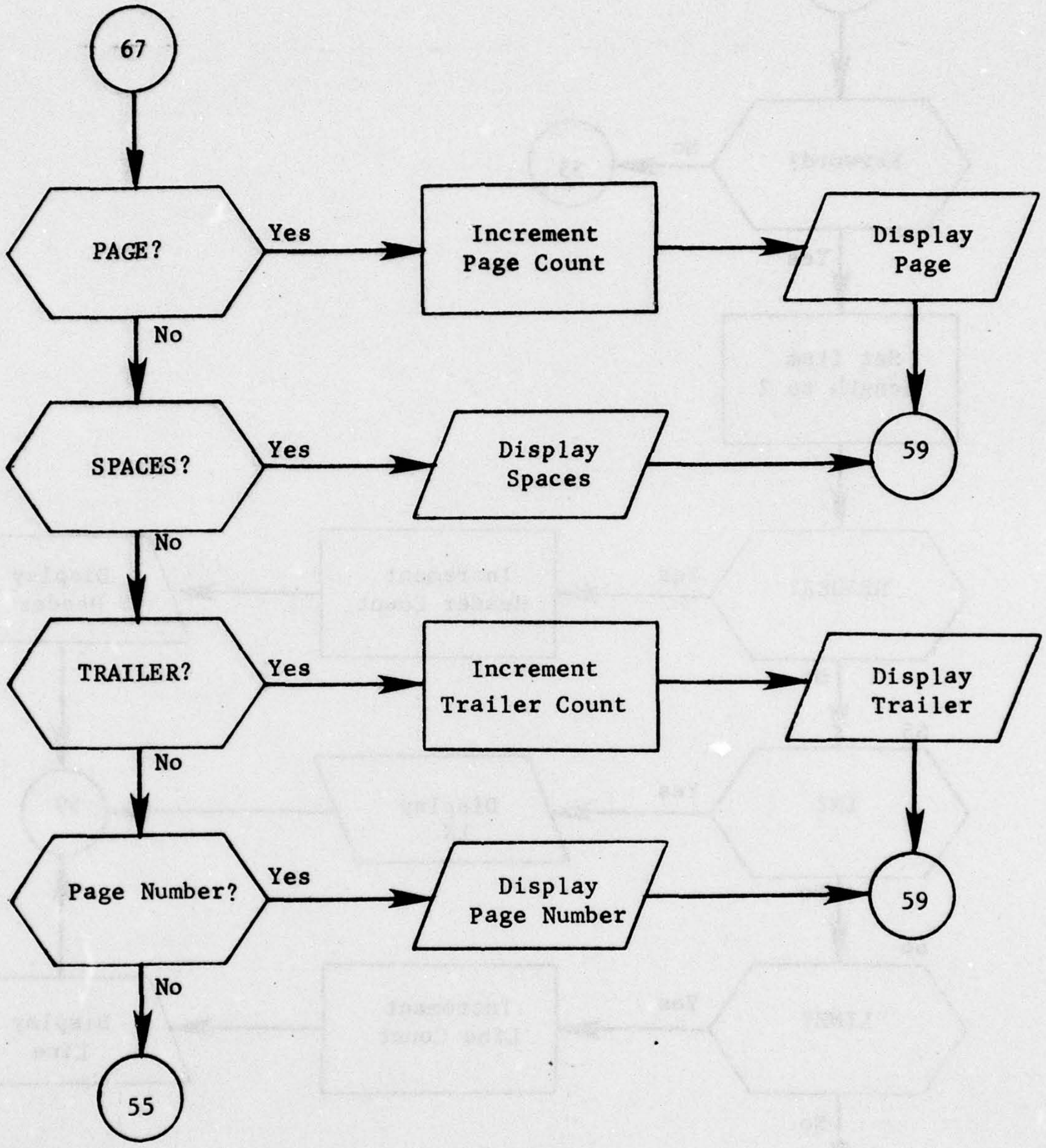


Figure 98. (Part 5 of 5)

6.10 Subroutine DSPMAK*

PURPOSE:	To create displays from tables built by ALTER or DESIGN
ENTRY POINTS:	DSPMAK
FORMAL PARAMETERS:	None
COMMON BLOCKS:	ATLST, C20, DEFNMZ, DSPFRM DSPHED, IDPT OOPS, ORDER, RTLST, SCHEME, SORSCH, ZEES
SUBROUTINES CALLED:	ATFNDR, DSPPUT, FORMAK, LINKUP, SETSCH, TABMNT, UNCODE
CALLED BY:	ENTMOD (REPORT)

Method:

Step One

All DEFINE clauses are retrieved from utility table 1 and scanned for attributes. Each attribute is added to a list of attributes (ATNUMB). Each DEFINE clause is also checked for errors (see figure 99).

Step Two

The WHERE clause is now retrieved from utility table 2 and scanned for attributes. Any attributes found are added to the attribute list (ATNUMB). If the CLASS and/or SIDE attributes are found, the values provided for them are saved to aid in retrieval scheme construction (see figure 100).

Step Three

The SORT clause is now retrieved from utility table 3 and scanned for attributes. Any attributes found are added to the attribute list (ATNUMB) and flagged as being part of the print/sort record (ATCLZ=3). The SORT clause is also error checked in the process (see figure 101).

Step Four

The FORMAT clause is now retrieved from utility table 4 and scanned for attributes. Any attributes found are added to the attribute list (ATNUMB) and flagged as being part of the print/sort record (ATCLZ=3) (see figure 102).

*Main routine of overlay link RPTDMK

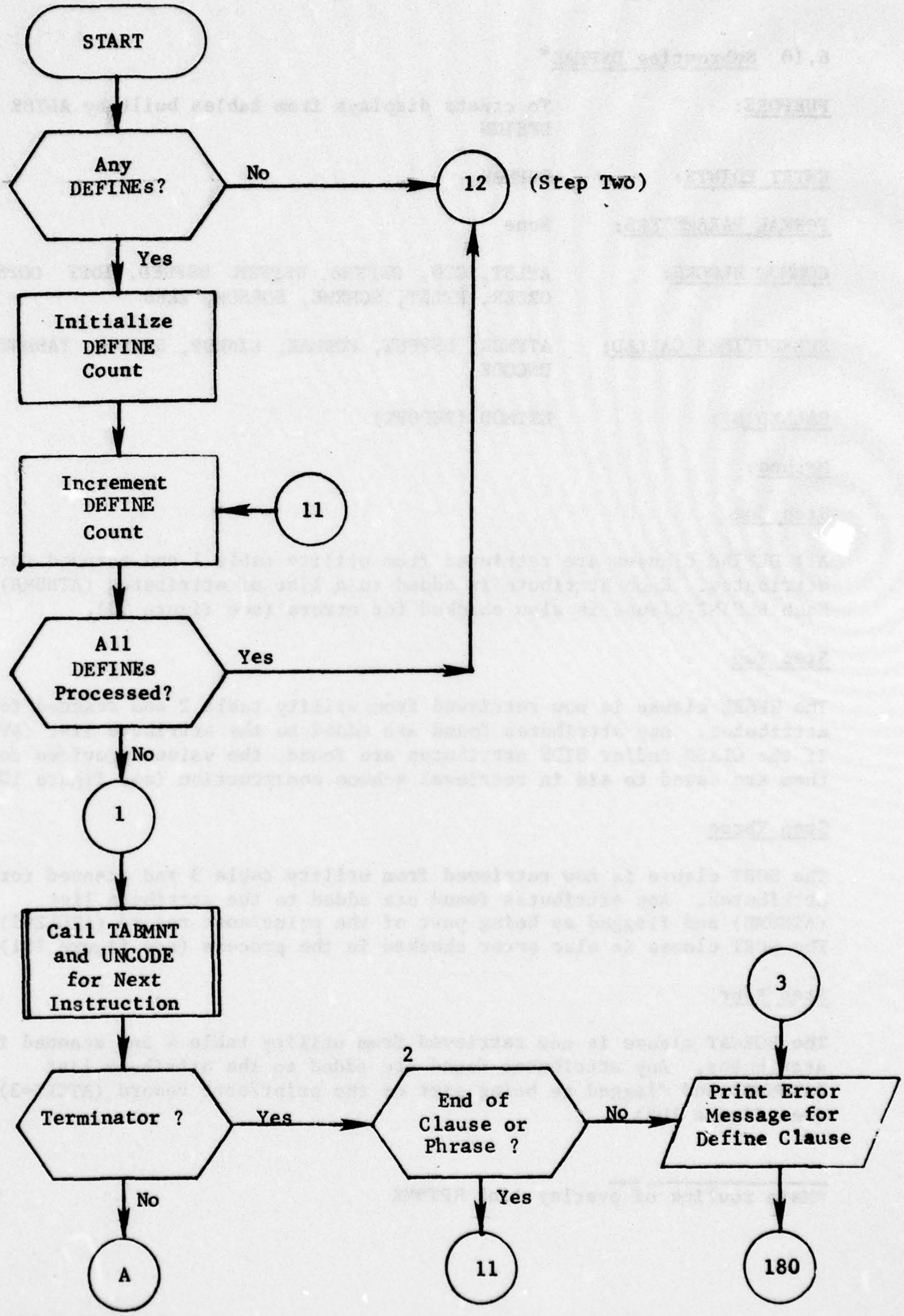


Figure 99. Subroutine DSPMAK: Step One (Part 1 of 2)

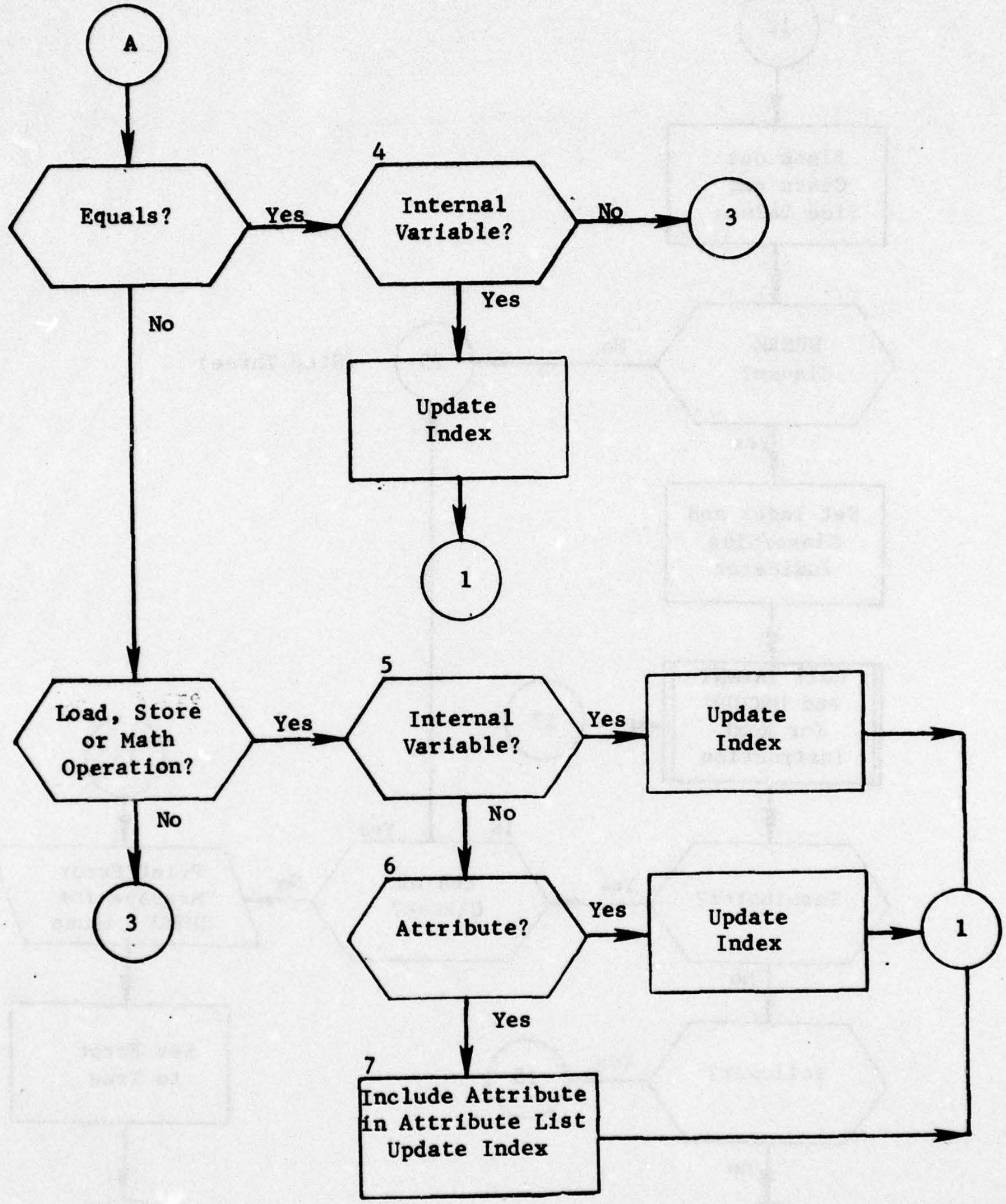


Figure 99. (Part 2 of 2)

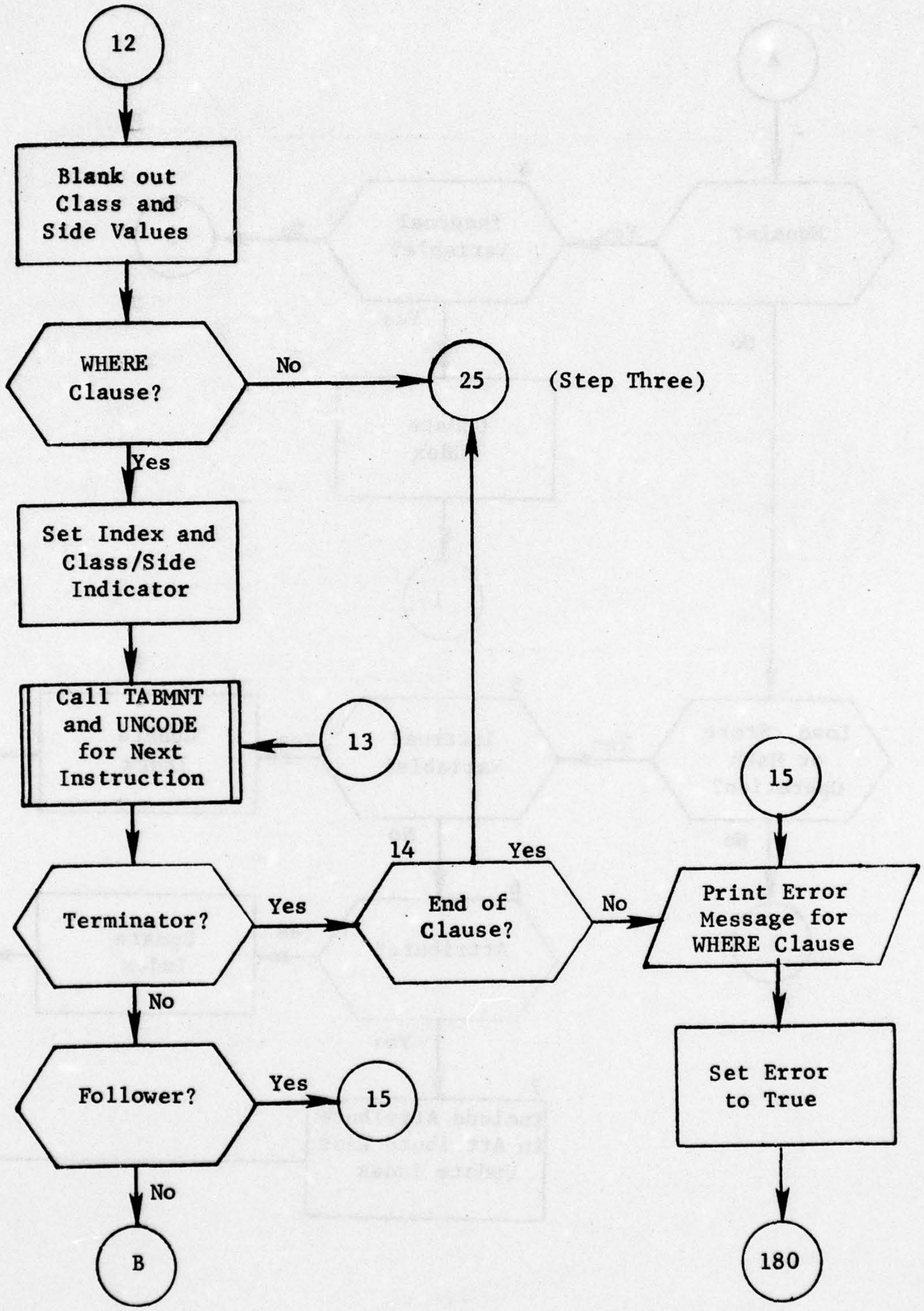


Figure 100. Subroutine DSPMAK: Step Two (Part 1 of 2)

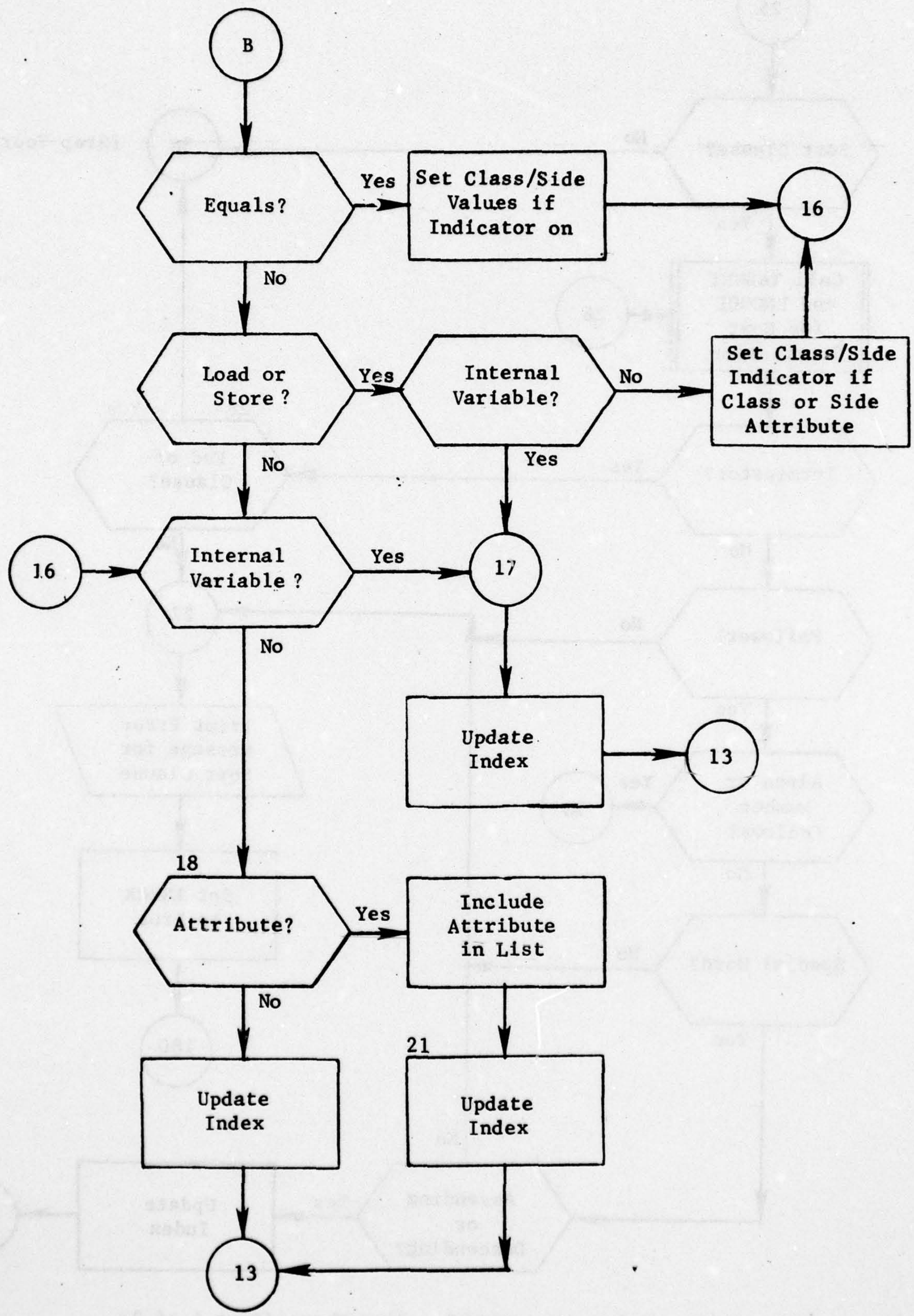


Figure 100. (Part 2 of 2)

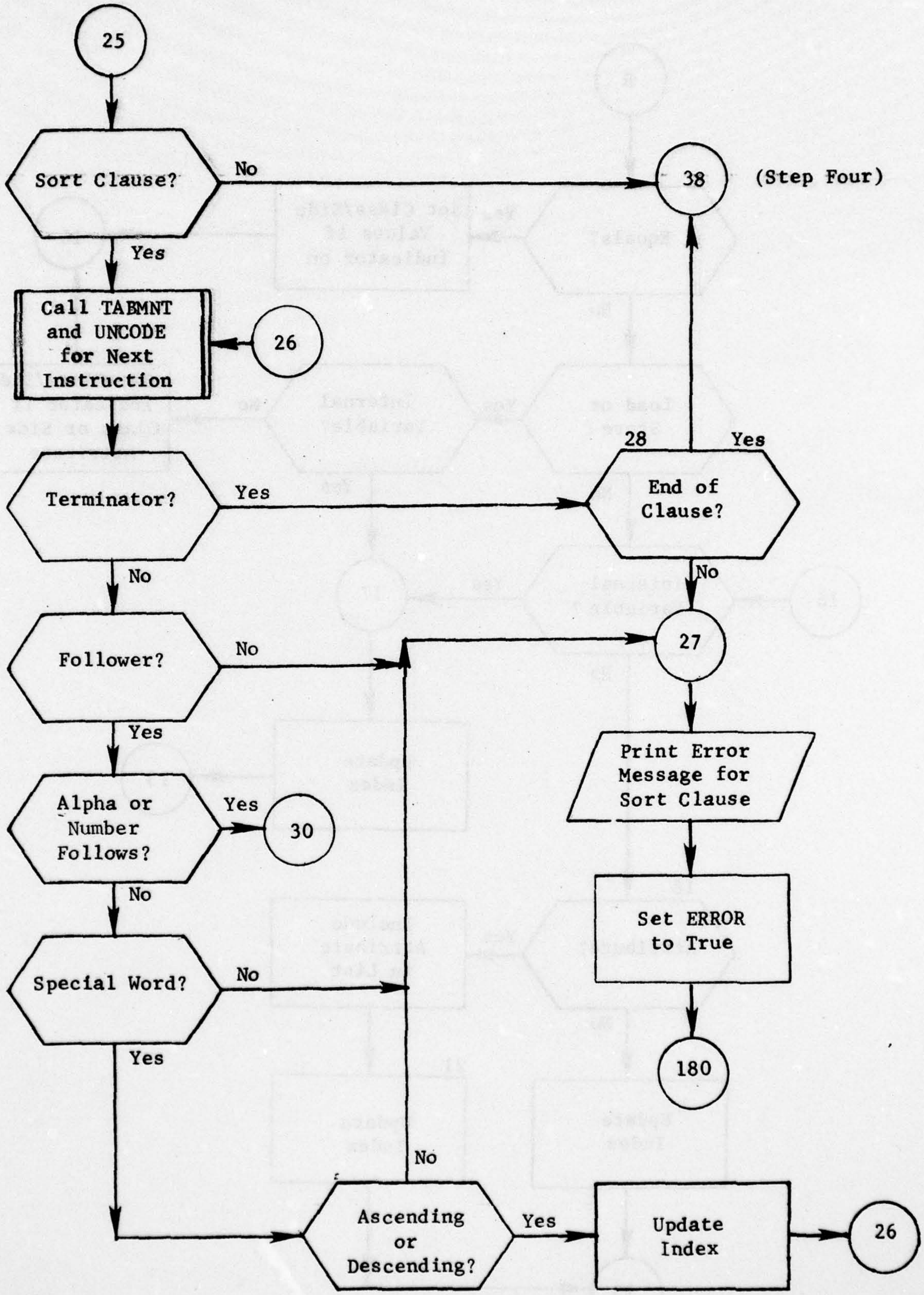


Figure 101. Subroutine DSPMAK: Step Three (Part 1 of 2)

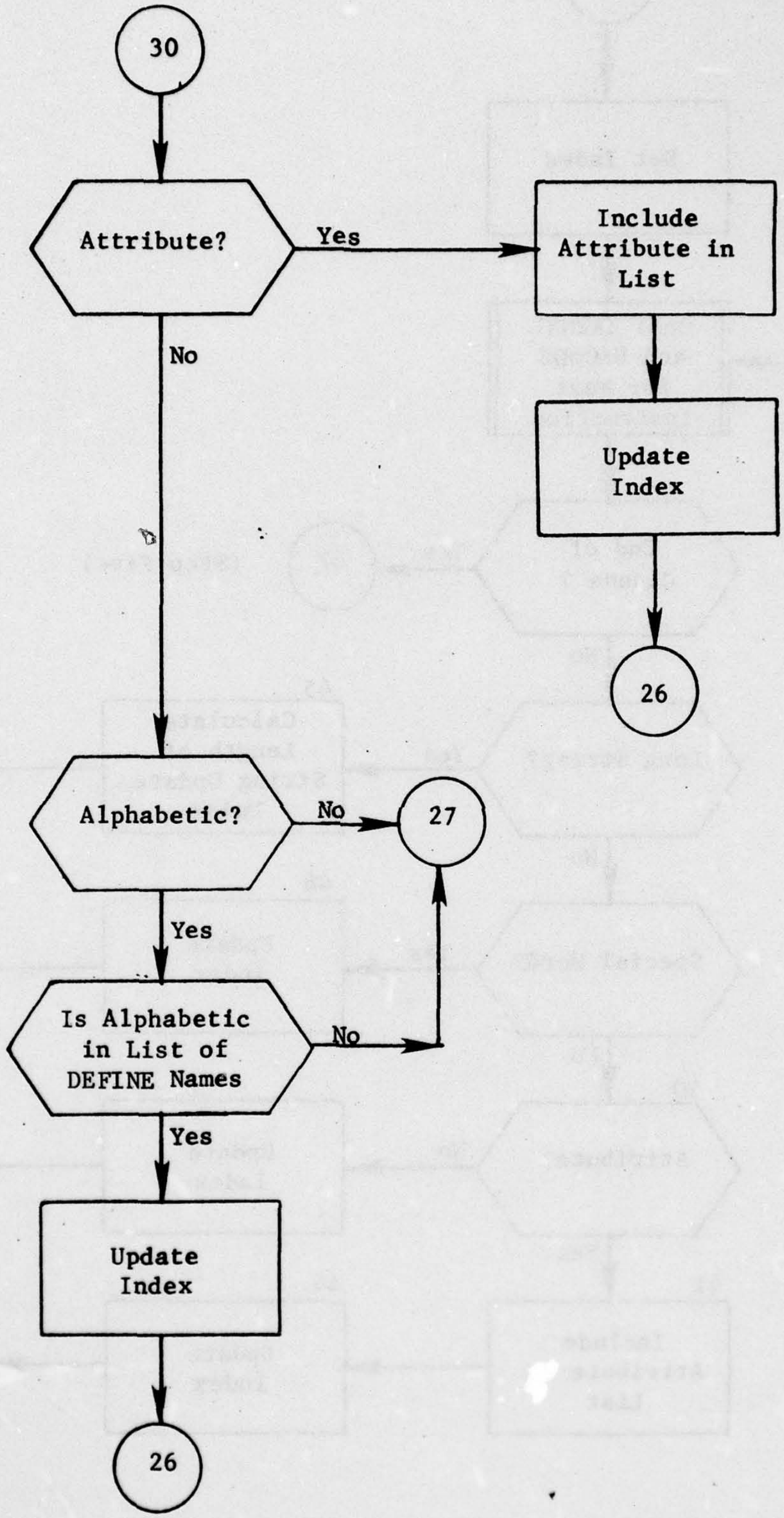


Figure 101. (Part 2 of 2)

AD-A054 310 COMMAND AND CONTROL TECHNICAL CENTER WASHINGTON D C F/G 15/7

THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK) PROGR--ETC(U)

JUN 77 D J SANDERS, P F MAYKRANTZ, J M HERRIN

UNCLASSIFIED CCTC-CSM-MM-9-77-V1-PT-2 SBIE-AD-E100 052 NL

2 OF 6

AD
A054310

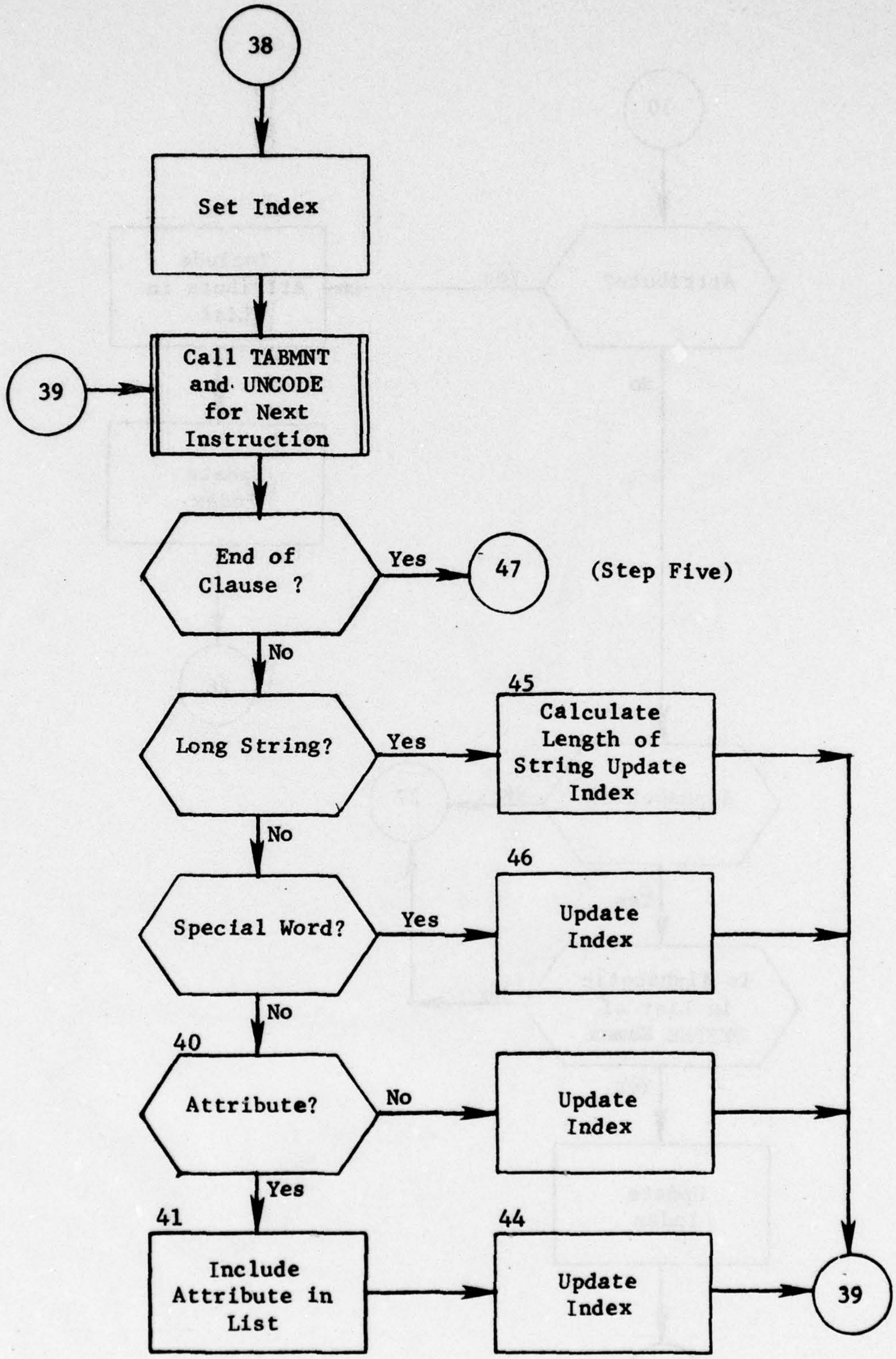


Figure 102. Subroutine DSPMAK: Step Four

Step Five

Now ATFNDR is called to build the record type list and subroutine LINKUP is called to complete the record type list and SETSCH is called to build the retrieval scheme. Next, each attribute flagged as being part of the print/sort record (ATCLZ=3) is added to a set of tables which contain its address and its assigned position in the print/sort record (elements 8 and 9 in table 18) (see figure 103).

Step Six

Now the set of DEFINE clauses is processed to build the DEFINE tables (elements 2-7 in table 18). The DEFINEs are placed in the tables in the order in which they should be executed. This order is dependent upon the presence within the execution instructions of other DEFINEs. The other DEFINEs have to be executed prior to the DEFINE which contains them. The process determines the mode (DFNMOD) and type (DFNTYP) of the DEFINE. 'Normal' DEFINES are also assigned positions in the Print/Sort record (DFNPOS). The alphabetic instructions which represents a DEFINE are altered in the following way:

- o The second word is changed from 9 to 8
- o The third word is set to the DEFINEs ordinal number
- o The fourth word is set to the DEFINEs mode

(If the DEFINE instruction being altered is one whose name is the same as the DEFINE clause which contains it (either a sum or product) the third and fourth words are set to zero.) (See figure 104.)

Step Seven

The WHERE clause is now scanned for alphabetic instructions that represent DEFINEs and these instructions are altered as per step six (see figure 105).

Step Eight

The SORT clause is now retrieved from utility table 3 in pairs of instructions. Each pair should contain an attribute or DEFINE name plus one of the special words: ASCENDING(A) or DESCENDING(D). The pairs are used to create the sort scheme in common block SORSCH (element 10 of table 18) (see figure 106).

Step Nine

The FORMAT clause is now retrieved from utility table 4 and used to create the print scheme (element 13 of table 18). The process begins with the first page (if a PAGE special word is not first it is inserted by the subroutine). Thereafter, for each PAGE, all HEADER items are

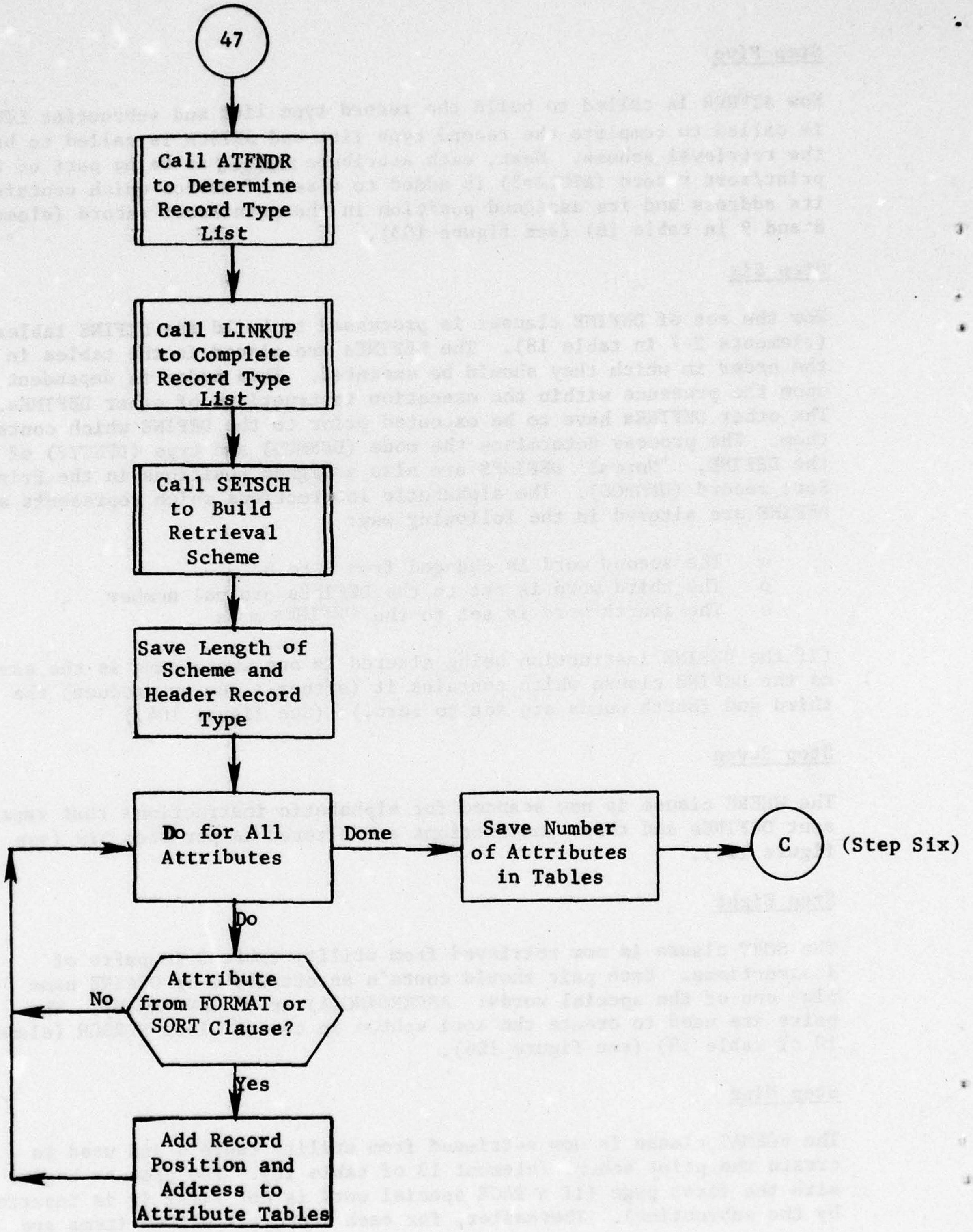


Figure 103. Subroutine DSPMAK: Step Five

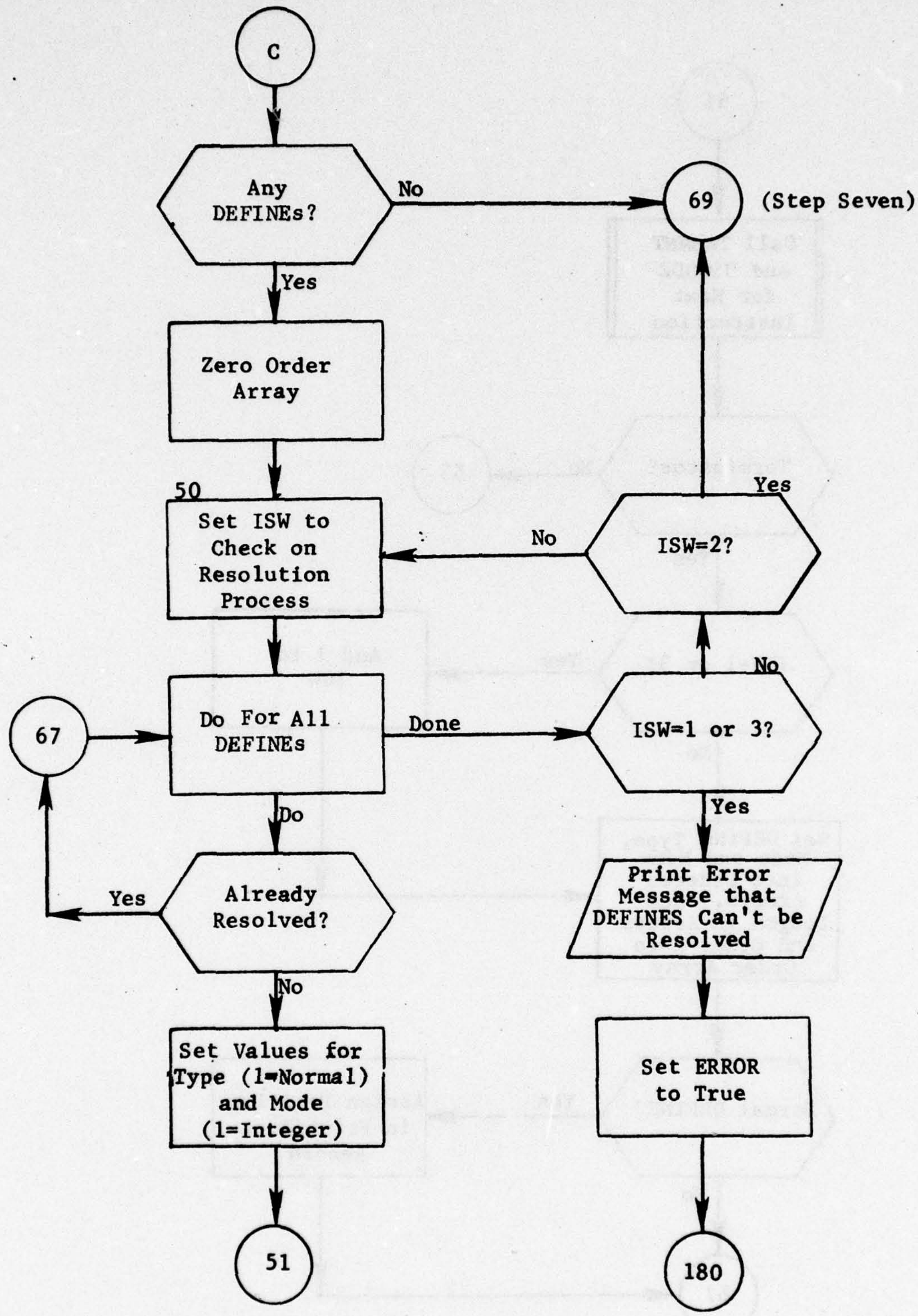


Figure 104. Subroutine DSPMAK: Step Six (Part 1 of 5)

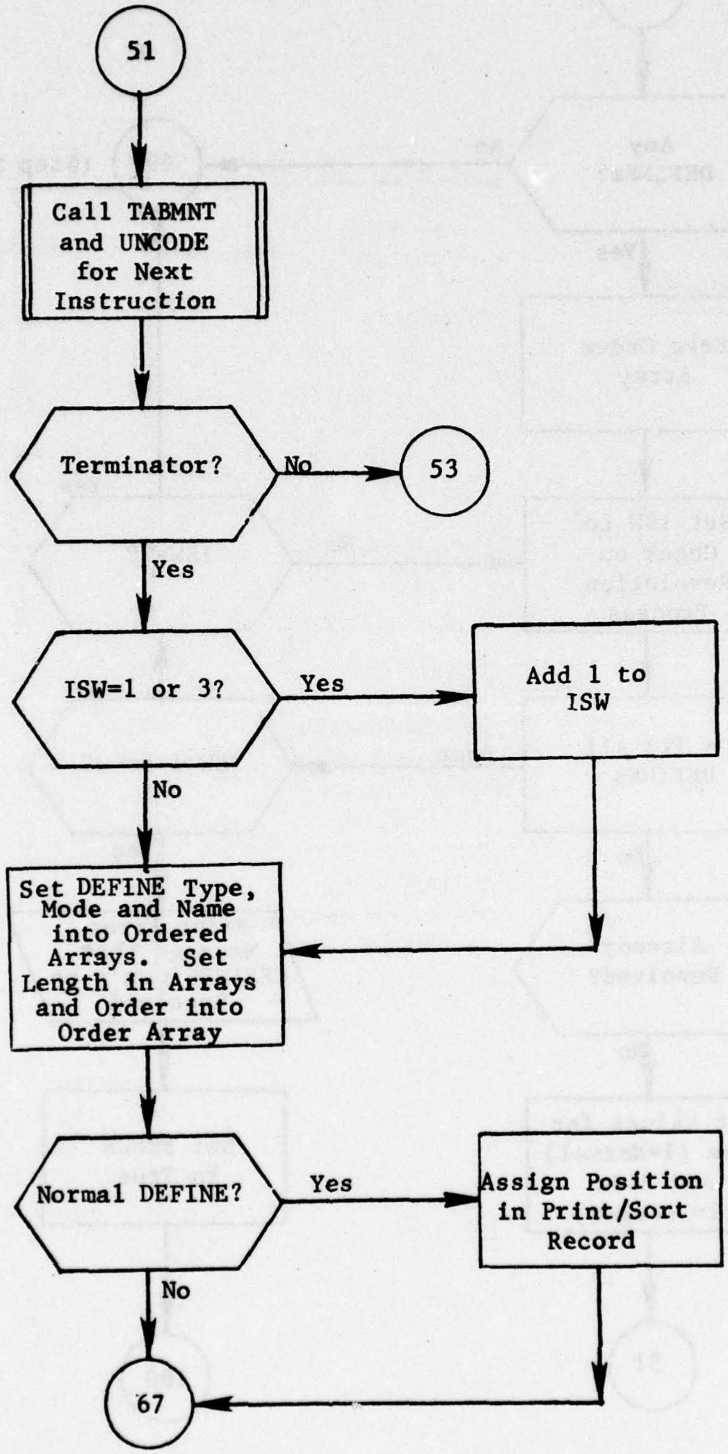


Figure 104. (Part 2 of 5)

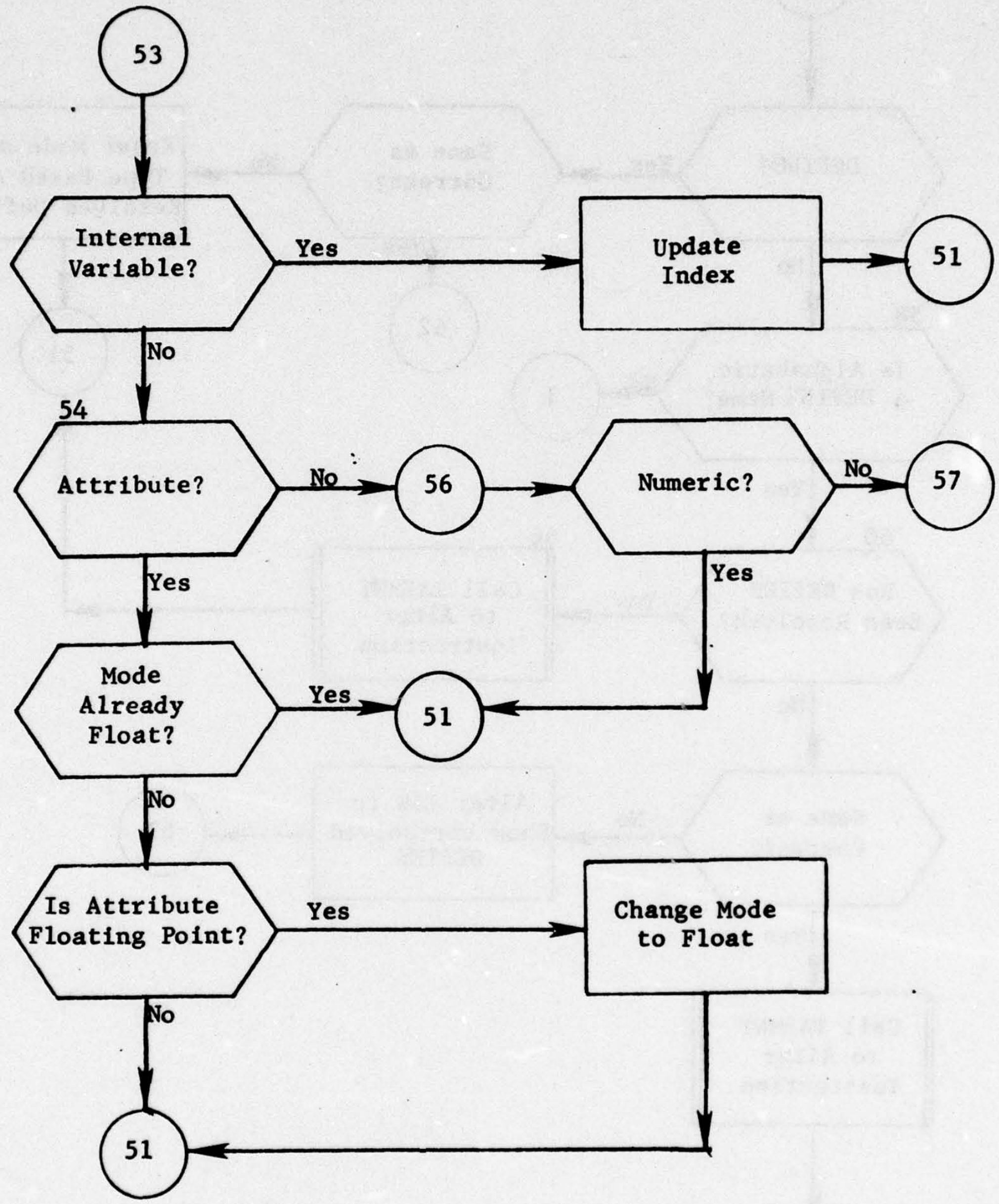


Figure 104. (Part 3 of 5)

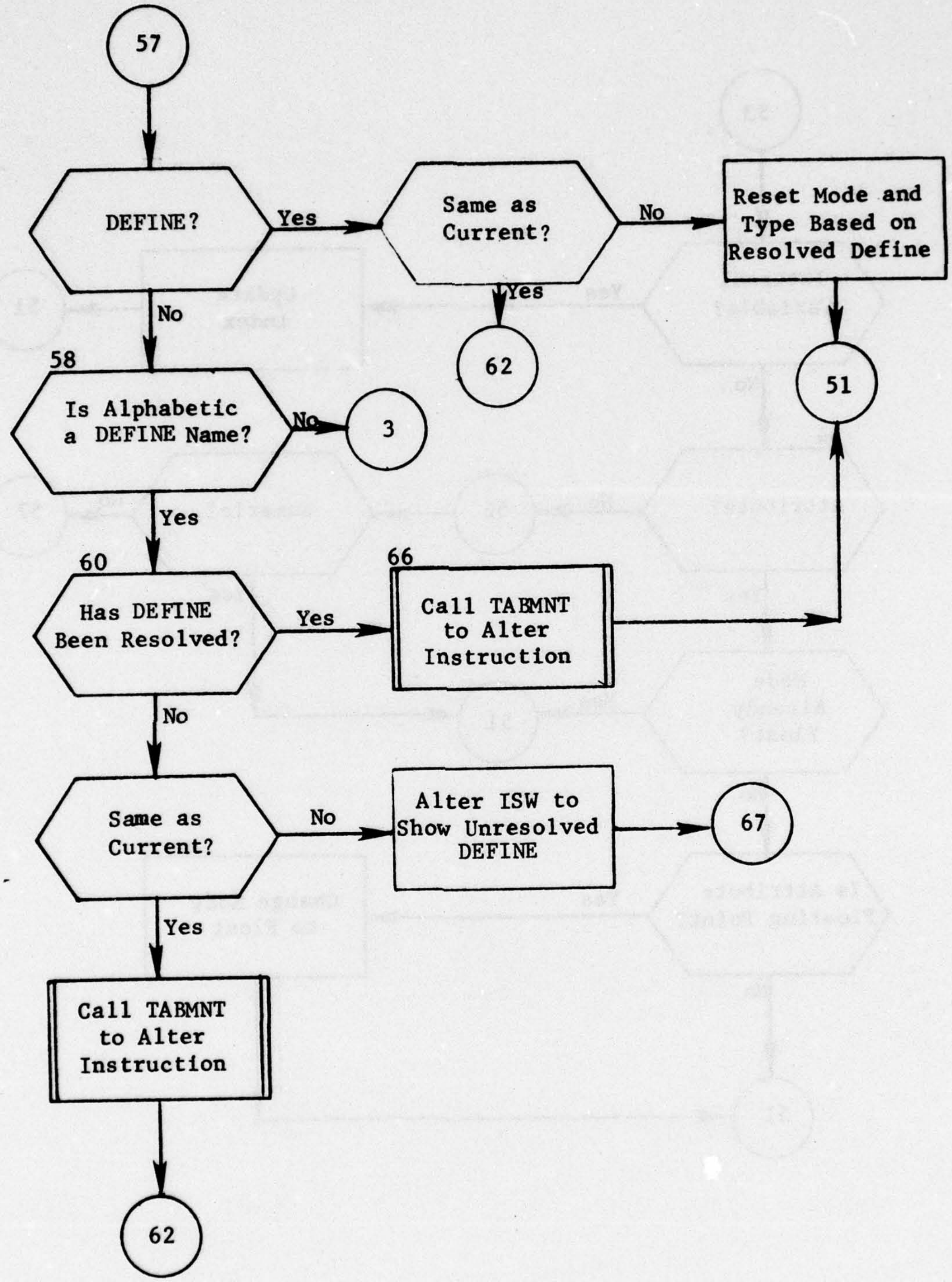


Figure 104. (Part 4 of 5)

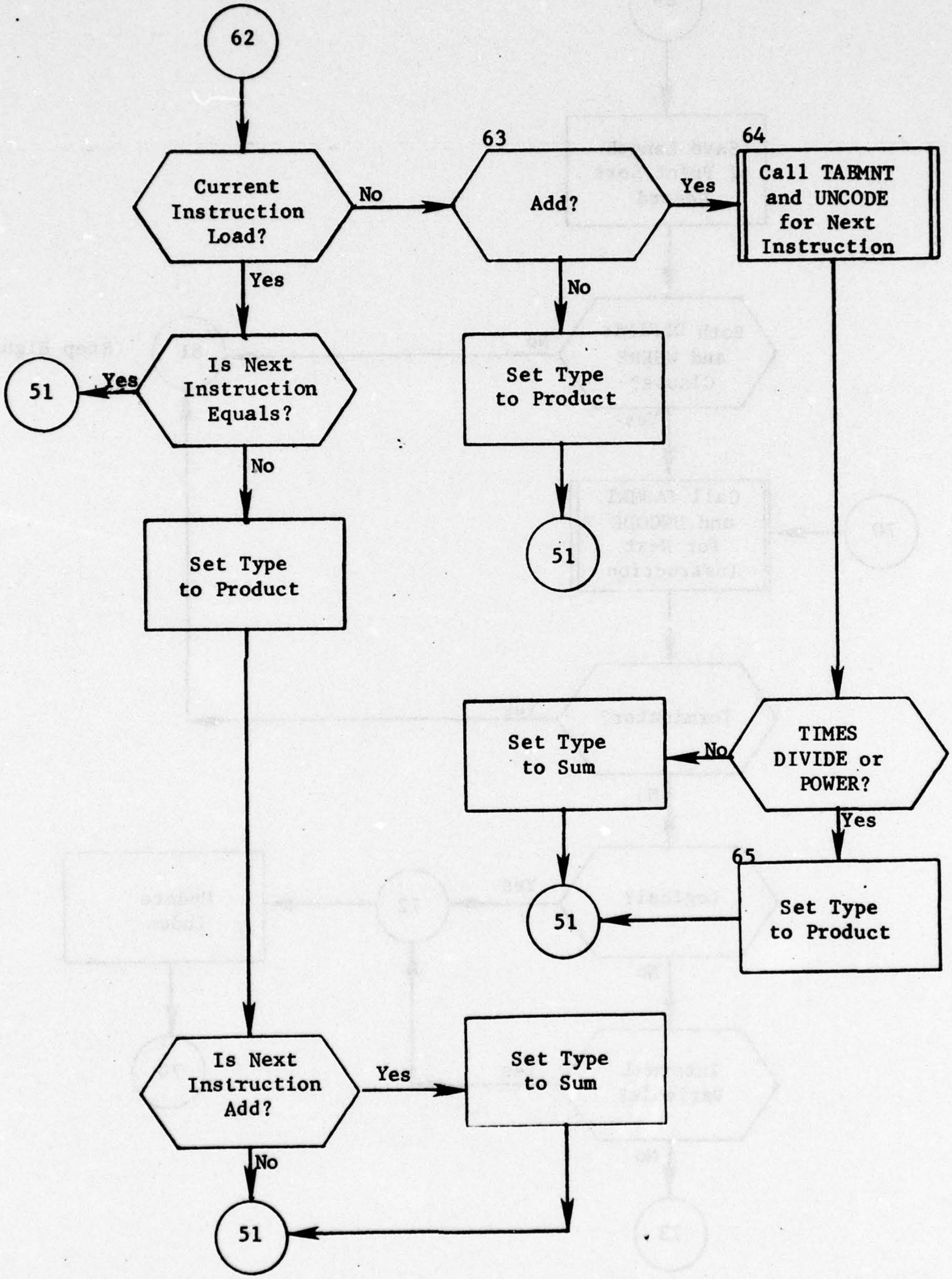


Figure 104. (Part 5 of 5)

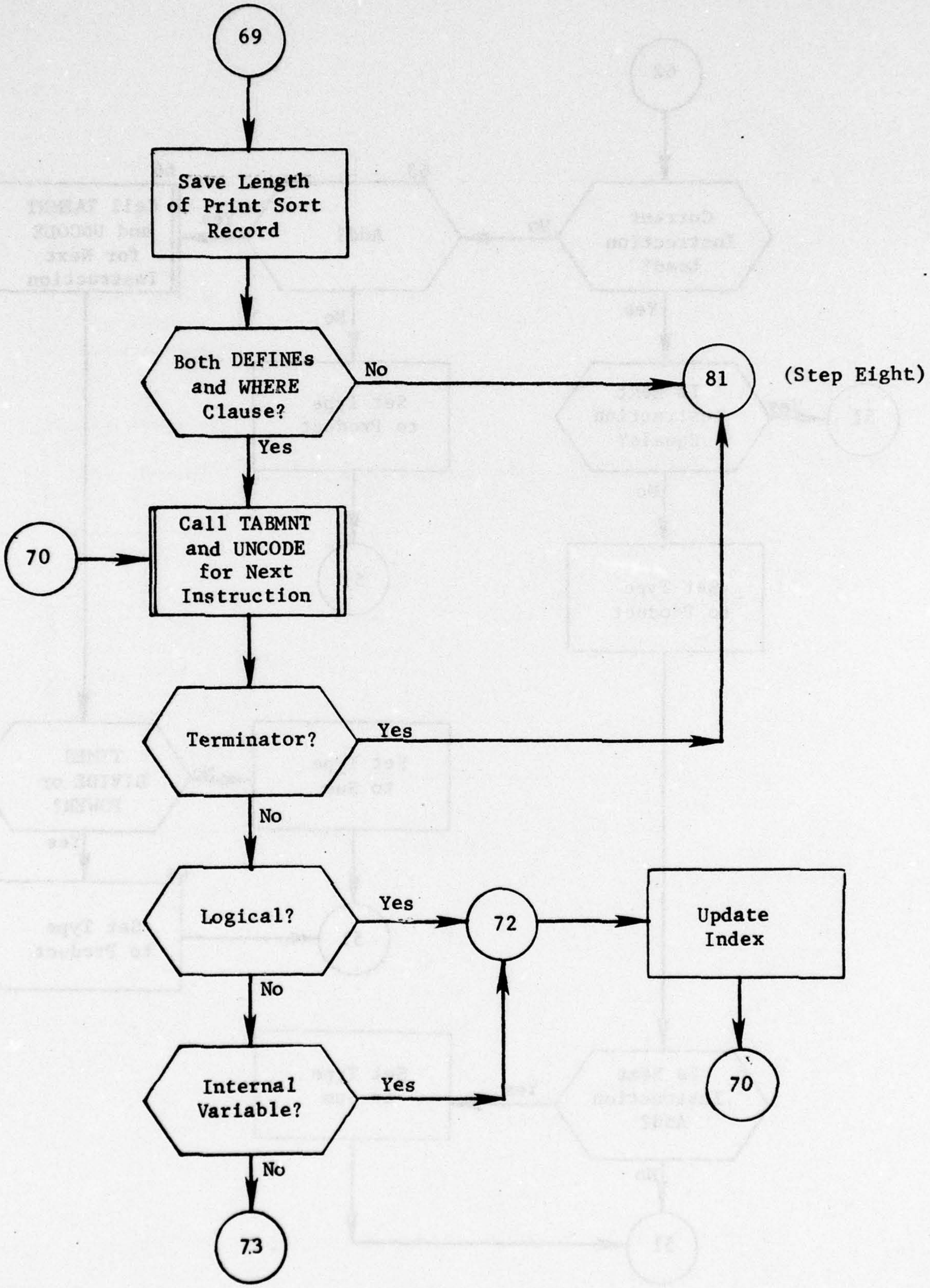


Figure 105. Subroutine DSPMAK: Step Seven (Part 1 of 2)

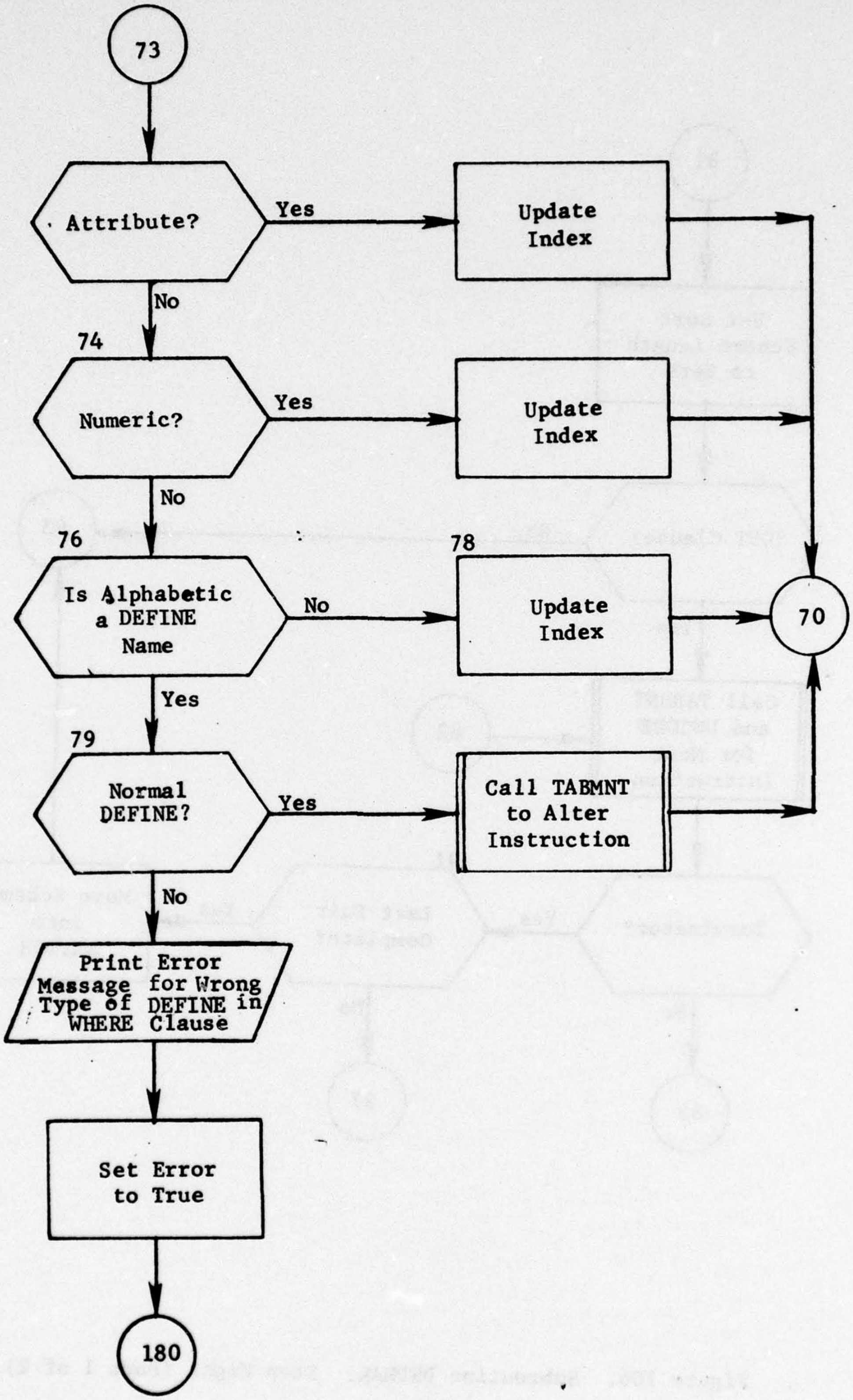


Figure 105. (Part 2 of 2)

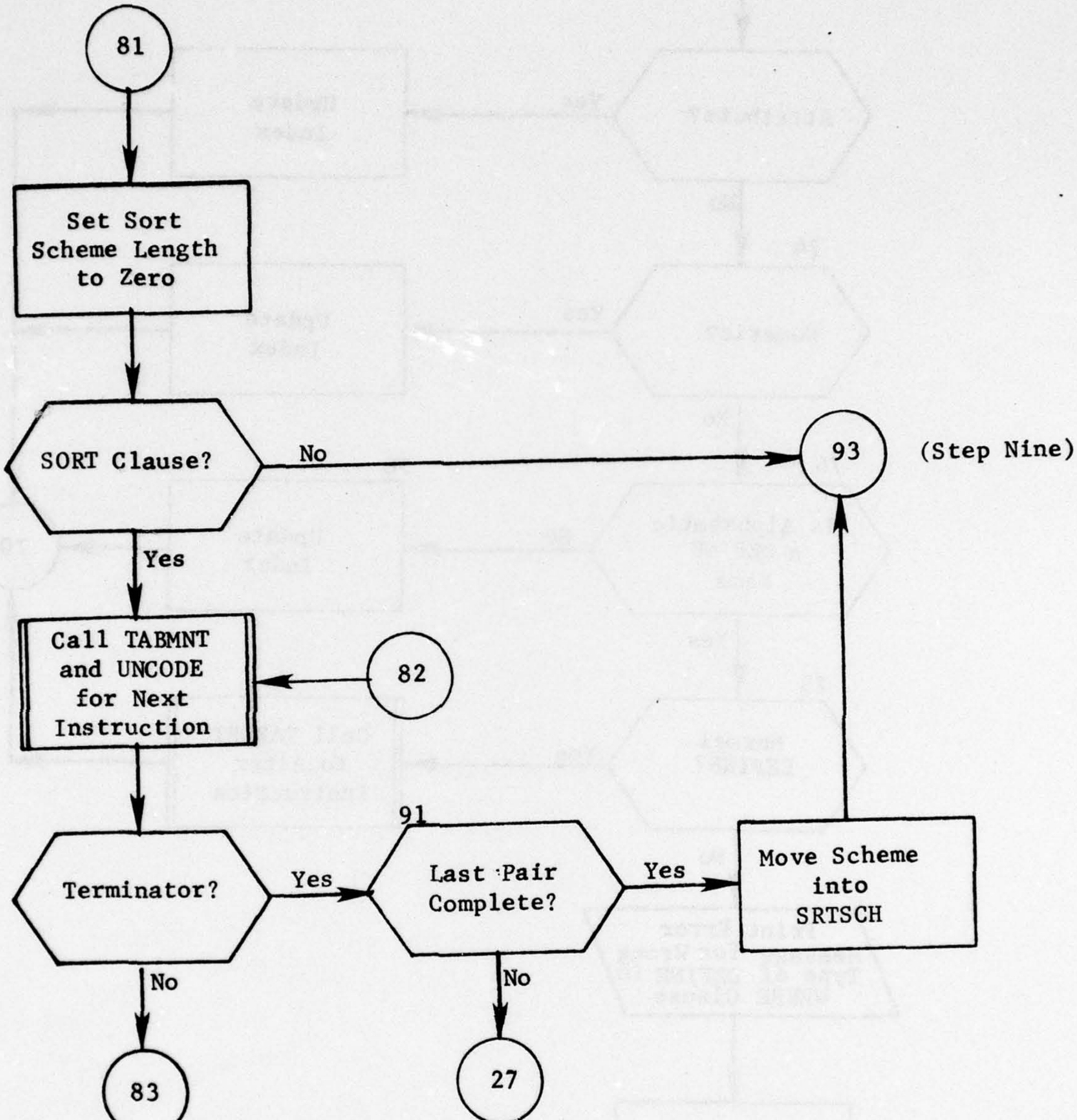


Figure 106. Subroutine DSPMAK: Step Eight (Part 1 of 2)

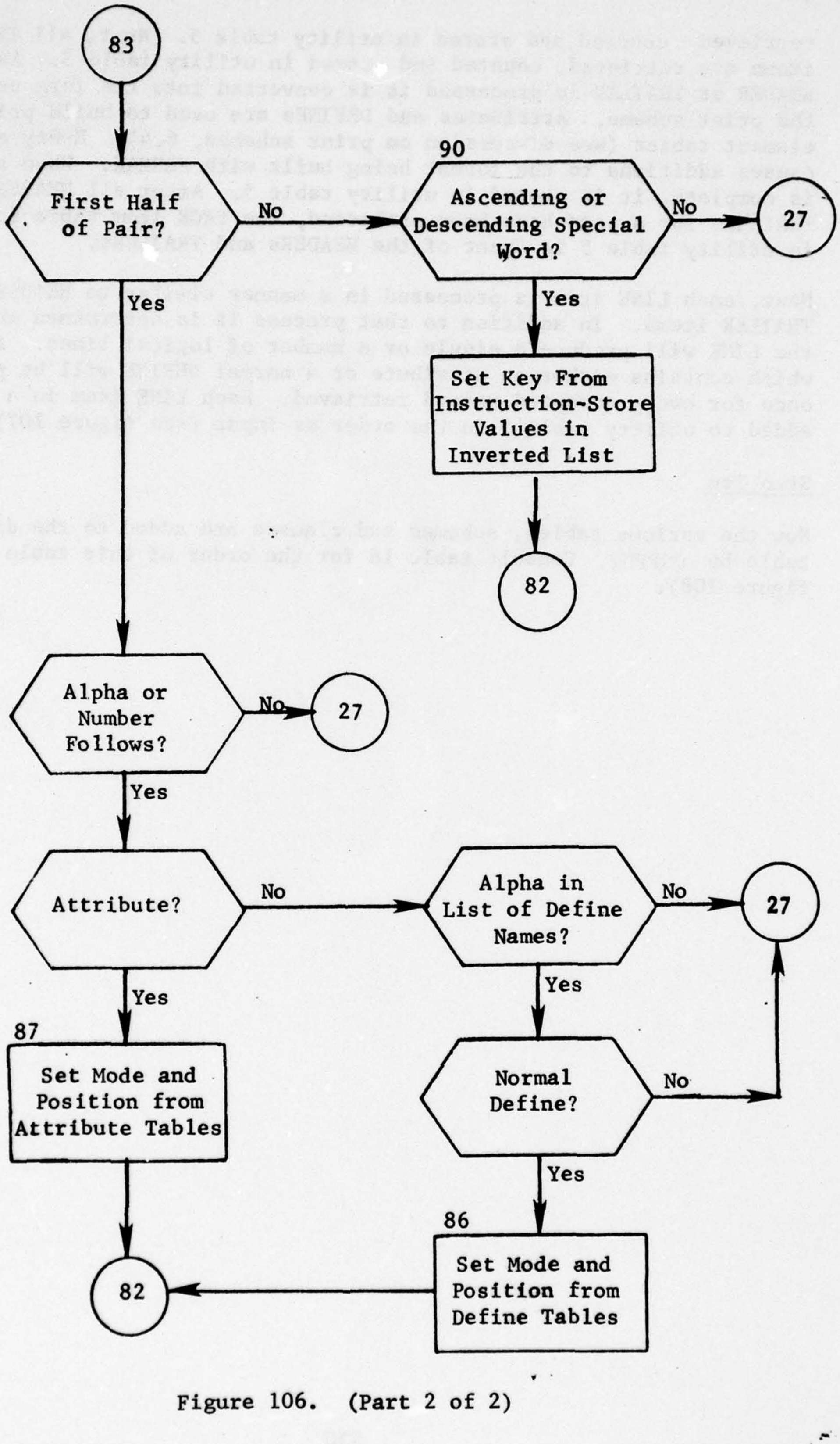


Figure 106. (Part 2 of 2)

retrieved, counted and stored in utility table 5. Next, all TRAILER items are retrieved, counted and stored in utility table 5. As each HEADER or TRAILER is processed it is converted into the form used in the print scheme. Attributes and DEFINEs are used to build print element tables (see discussion on print schemes, 6.4). Every element causes additions to the format being built with FORMAK. When an item is complete, it is stored in utility table 5. After all HEADERS and TRAILERs for a page have been processed, the PAGE item table is inserted in utility table 5 in front of the HEADERs and TRAILERs.

Next, each LINE item is processed in a manner similar to HEADER and TRAILER items. In addition to that process it is determined whether the LINE will produce a single or a number of logical lines. Any LINE which contains either an attribute or a normal DEFINE will be produced once for every accepted record retrieved. Each LINE item in a PAGE is added to utility table 5 in the order as input (see figure 107).

Step Ten

Now the various tables, schemes and clauses are added to the display table by DSPPUT. Consult table 18 for the order of this table (see figure 108).

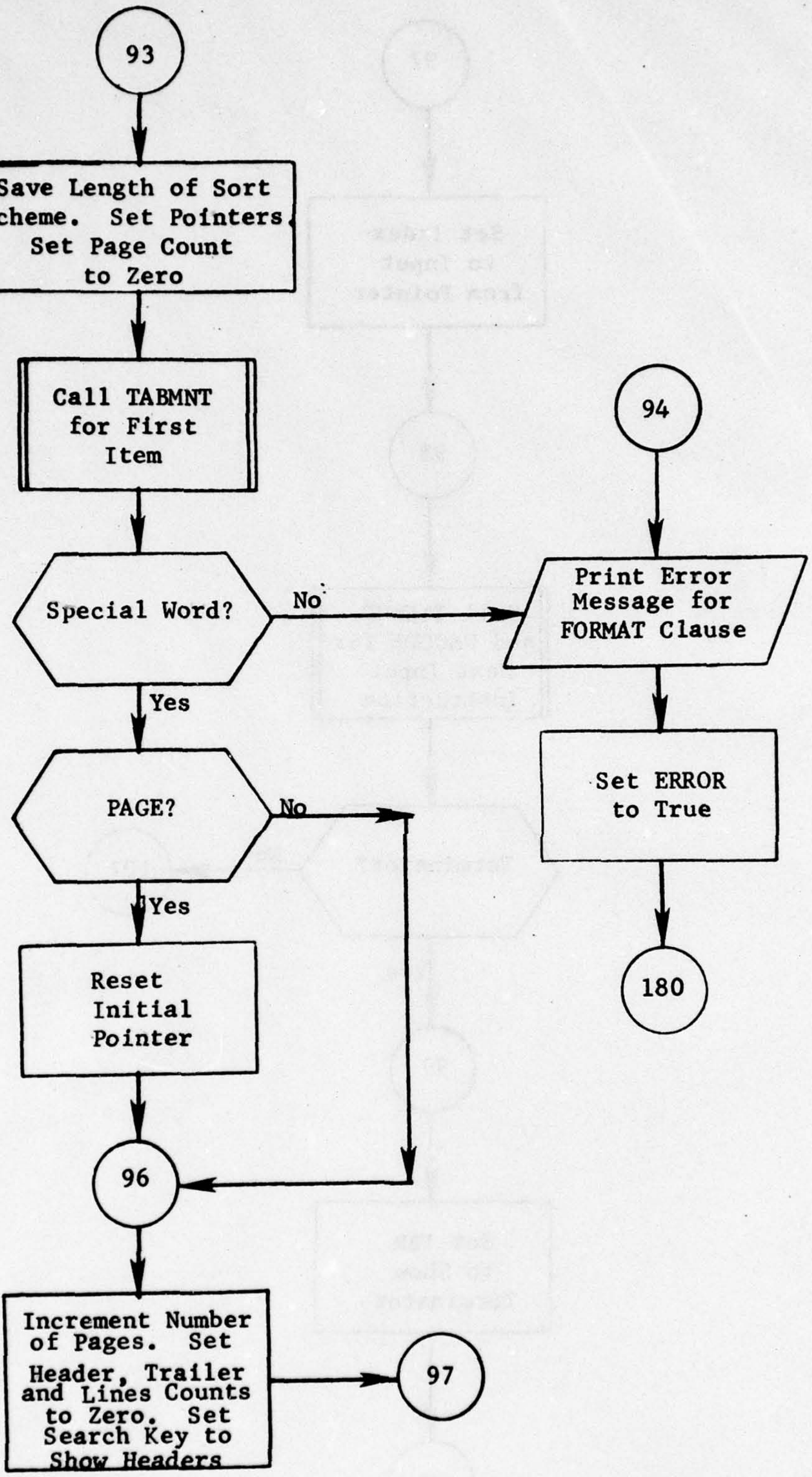


Figure 107. Subroutine DSPMAK: Step Nine (Part 1 of 12)

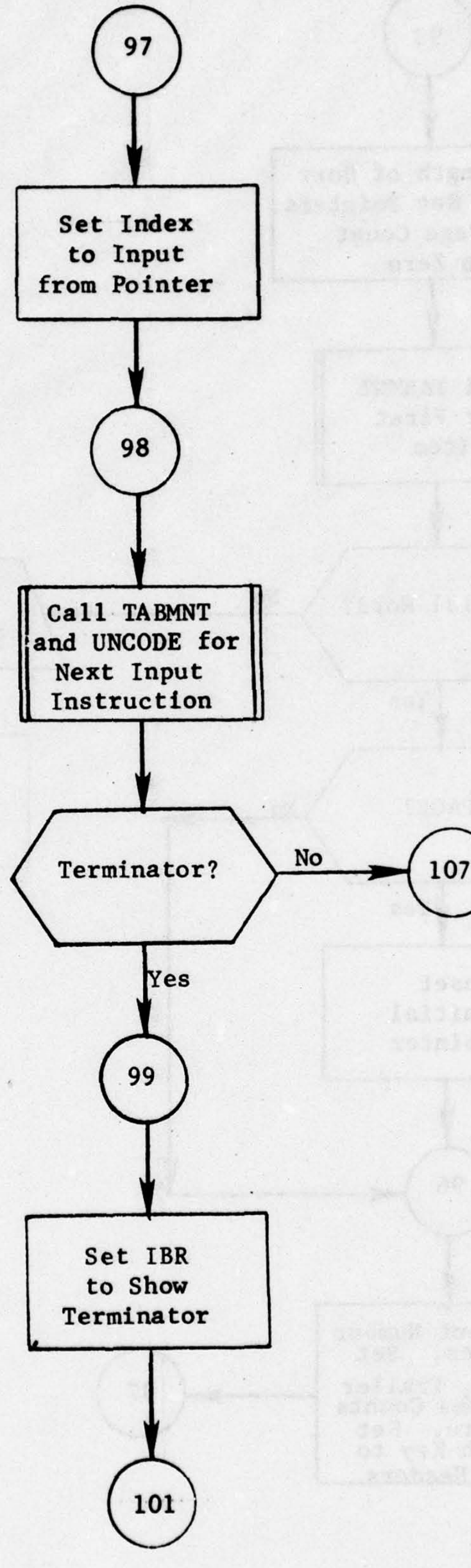


Figure 107. (Part 2 of 12)

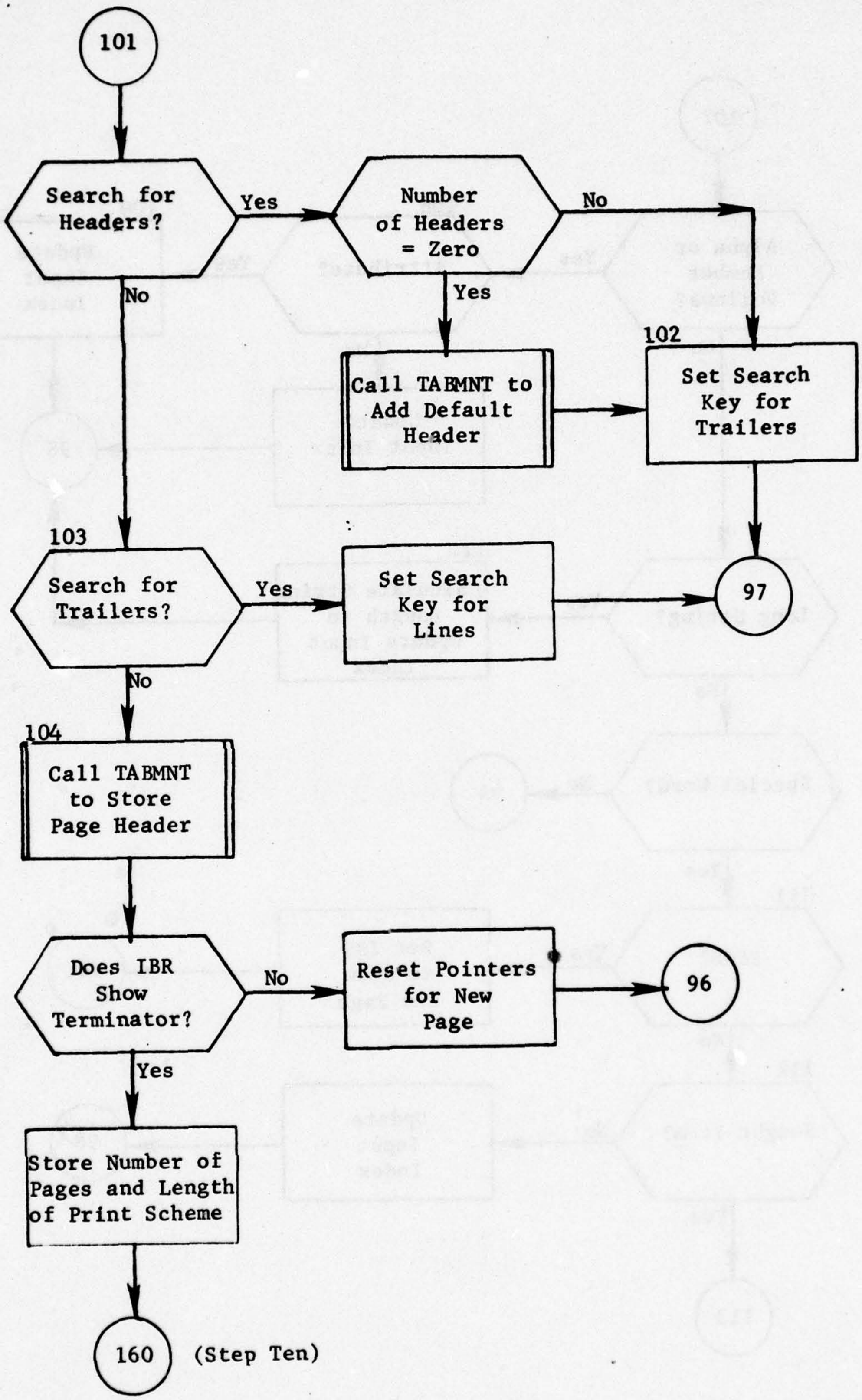


Figure 107. (Part 3 of 12)

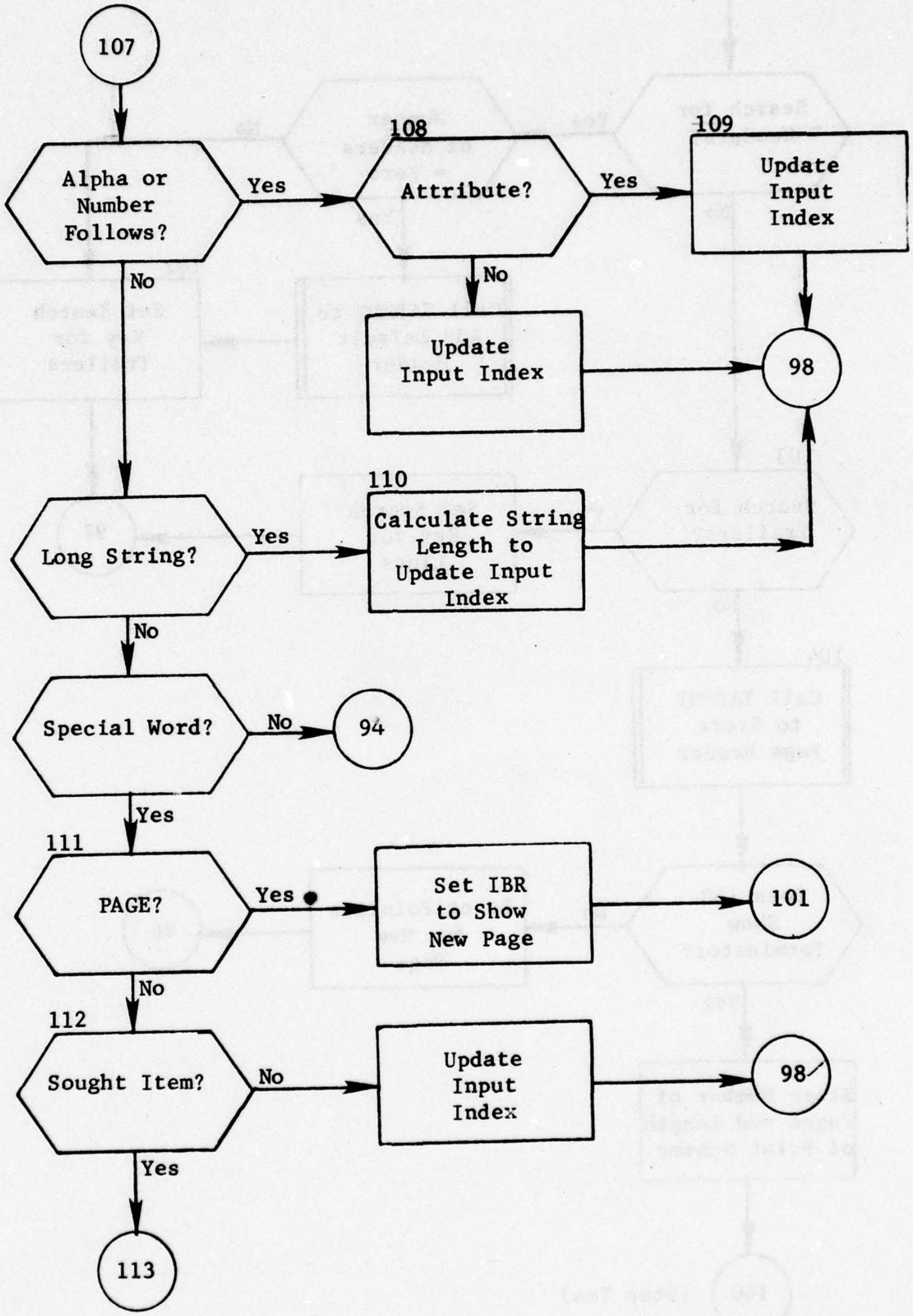


Figure 107. (Part 4 of 12)

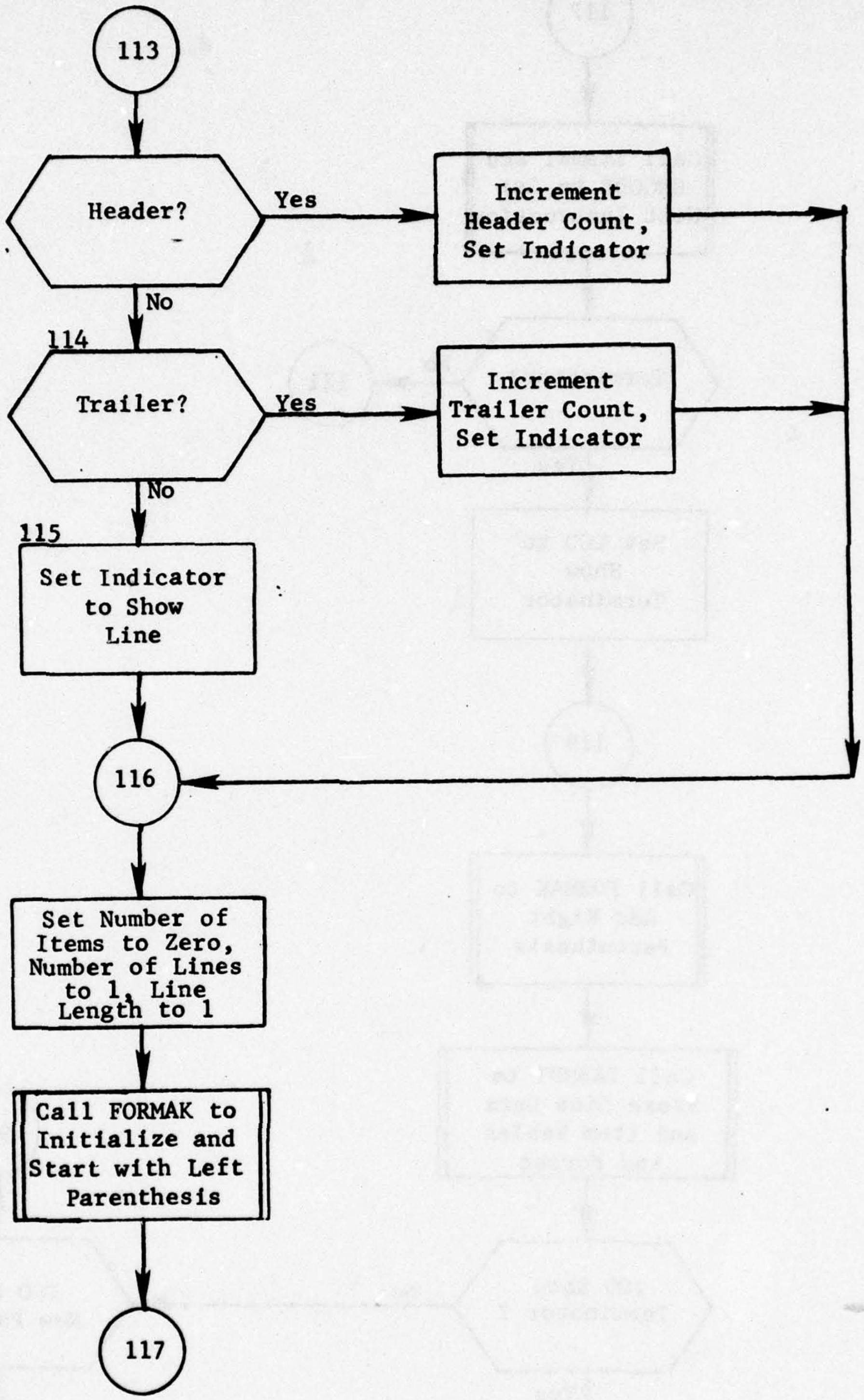


Figure 107. (Part 5 of 12)

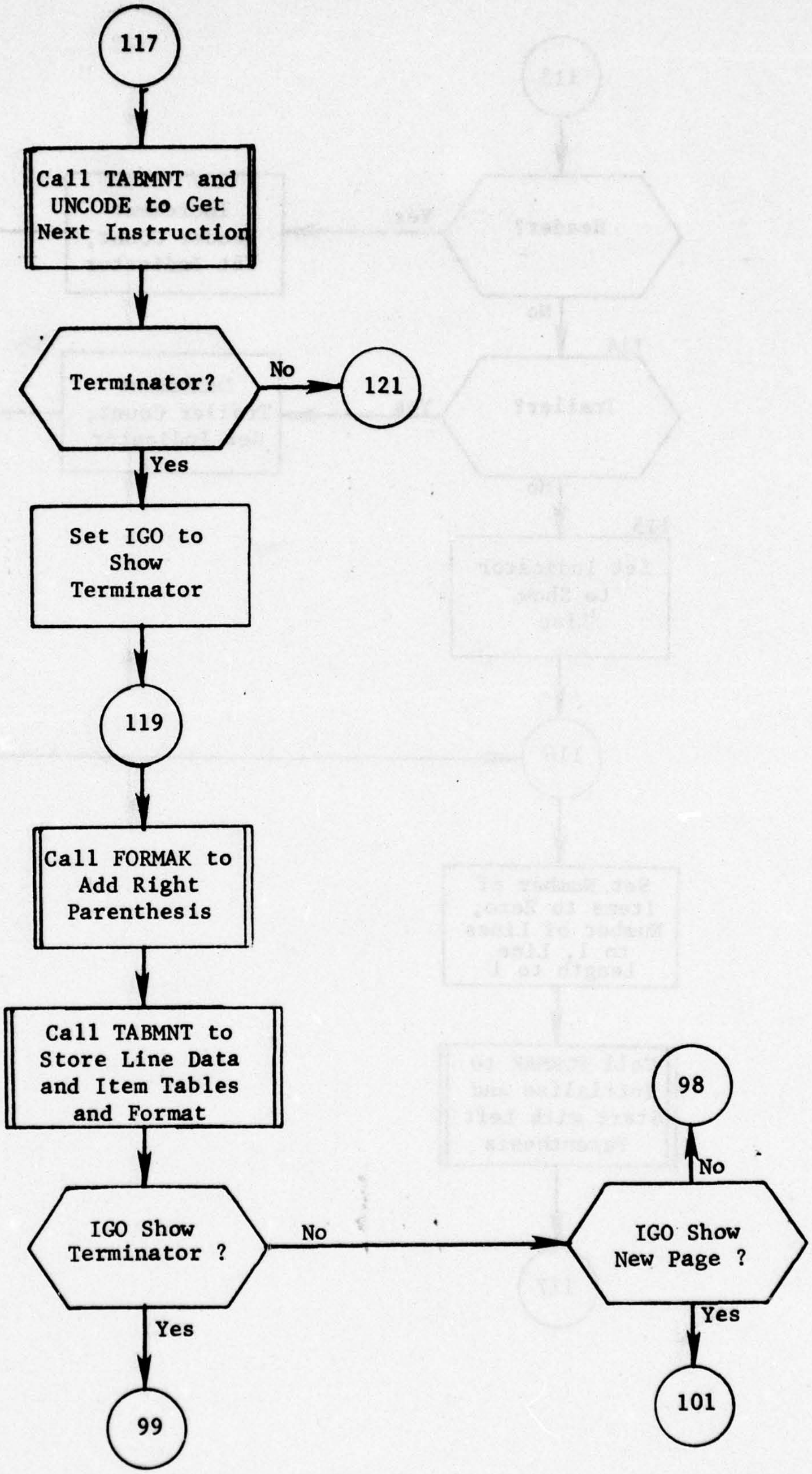


Figure 107. (Part 6 of 12)

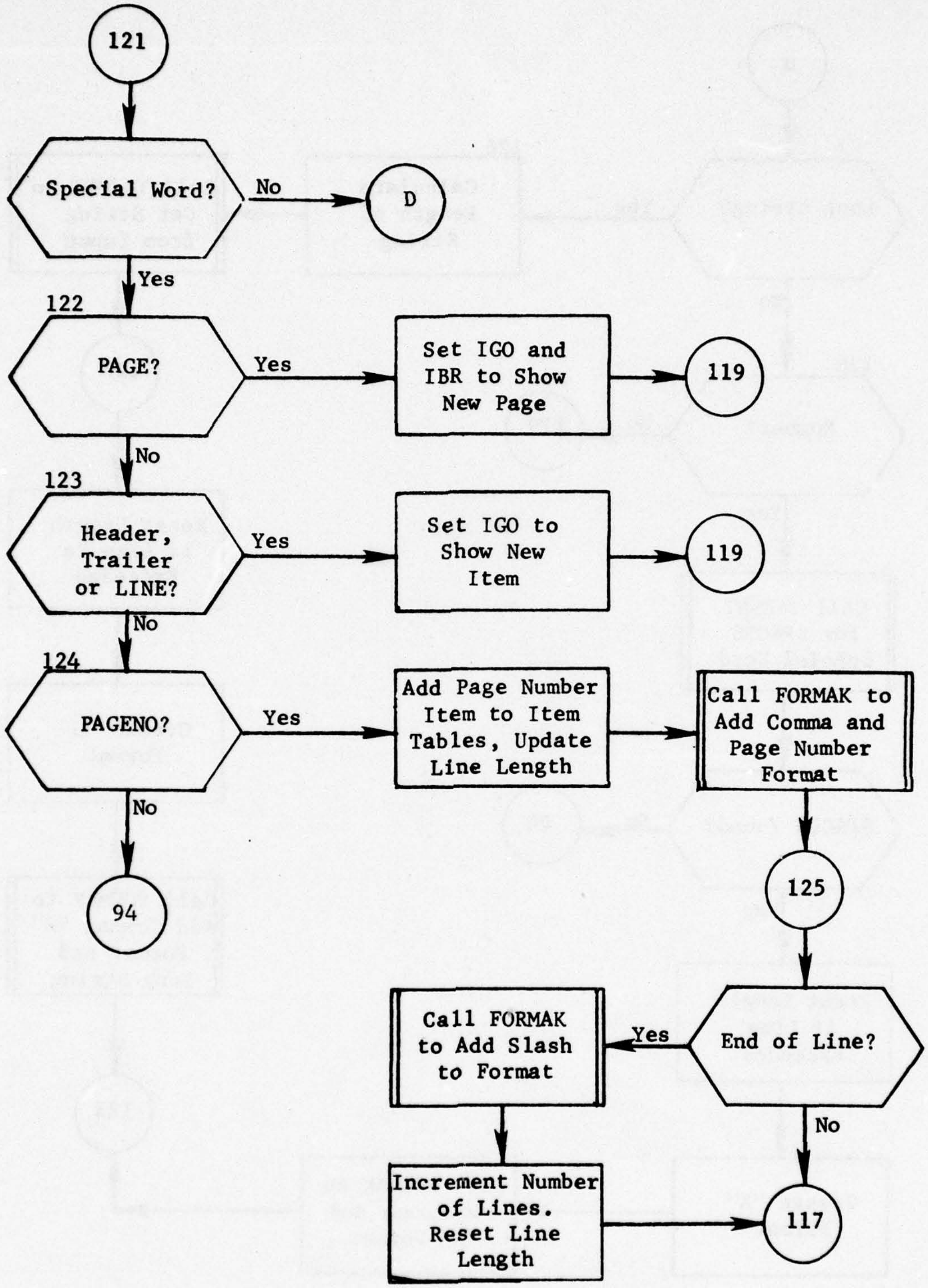


Figure 107. (Part 7 of 12)

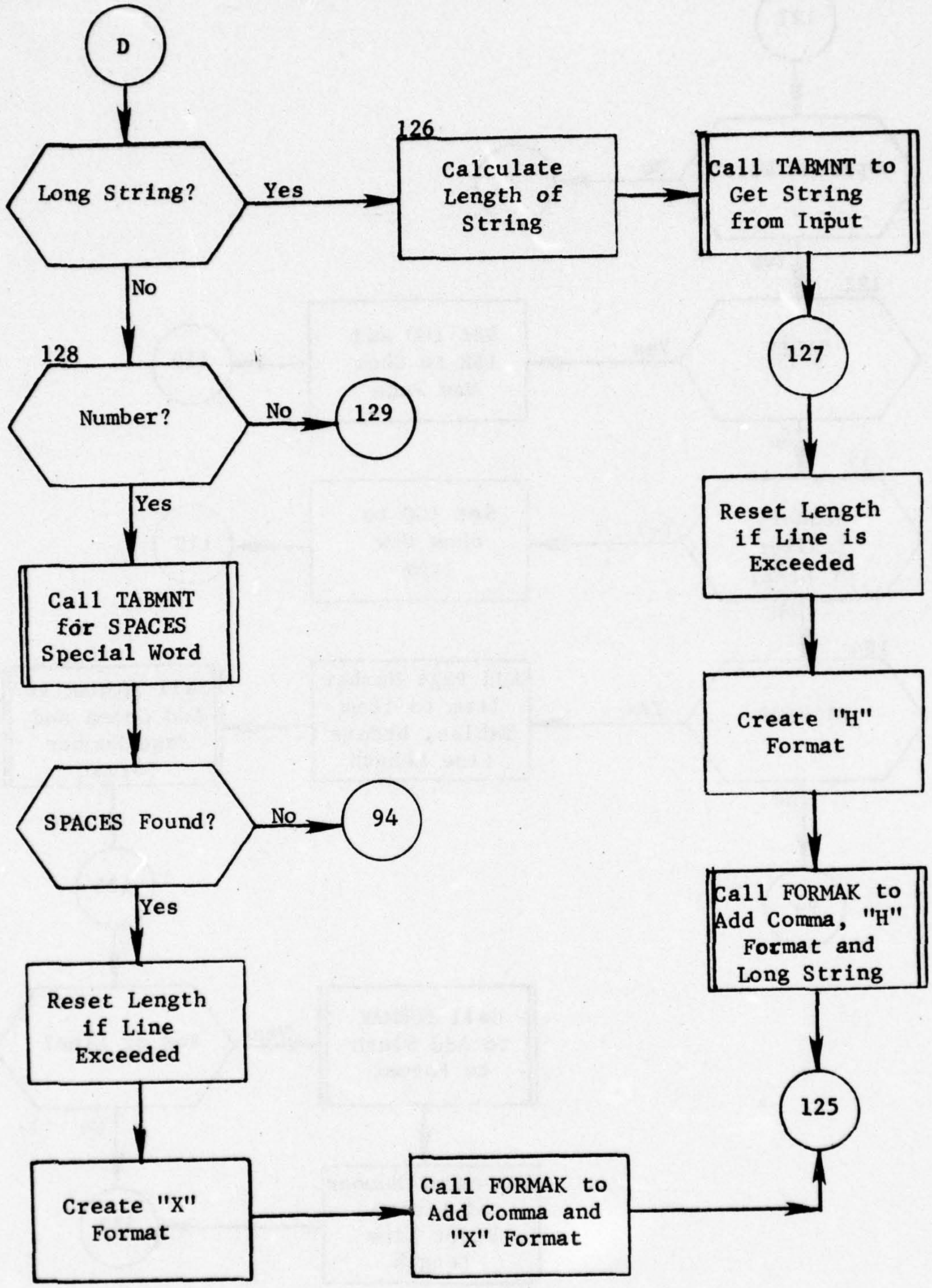


Figure 107. (Part 8 of 12)

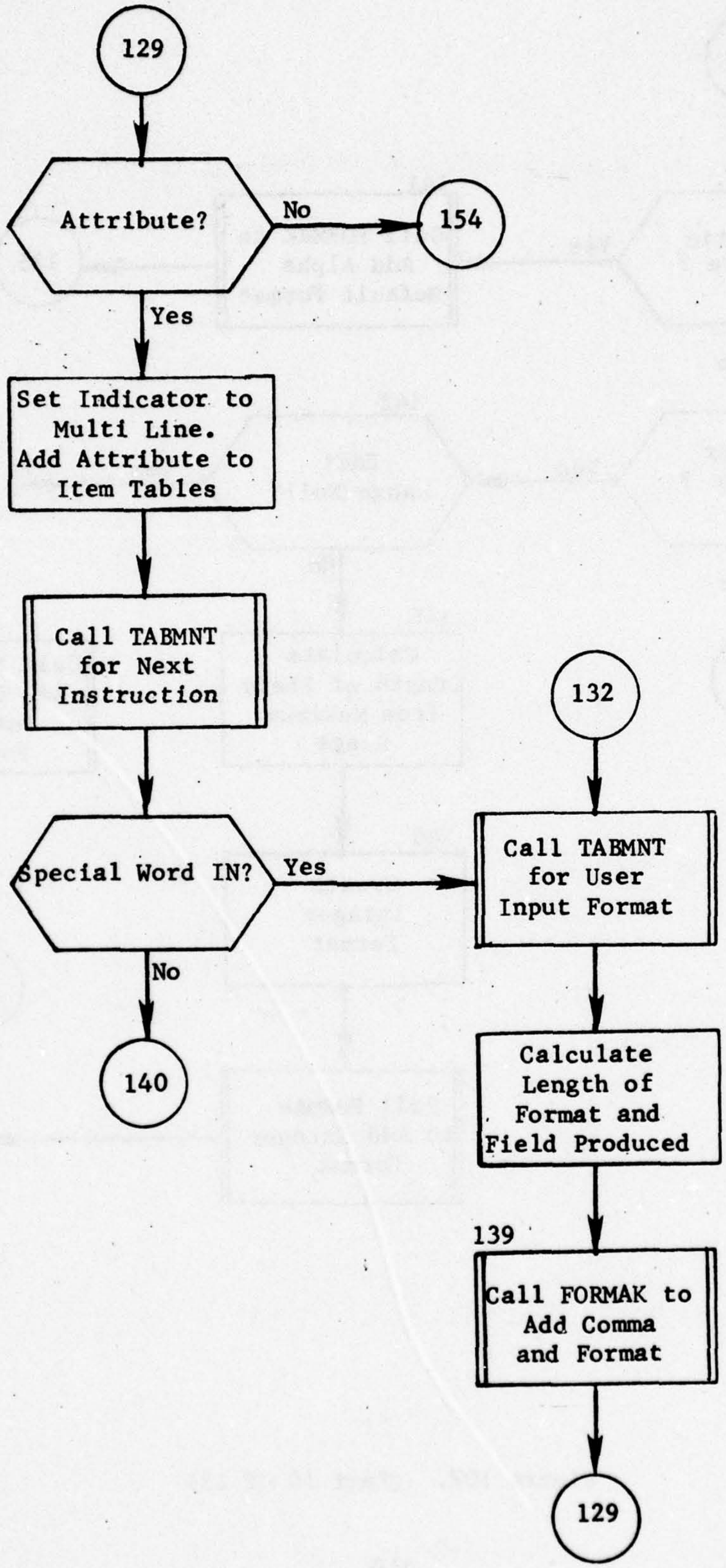


Figure 107. (Part 9 of 12)

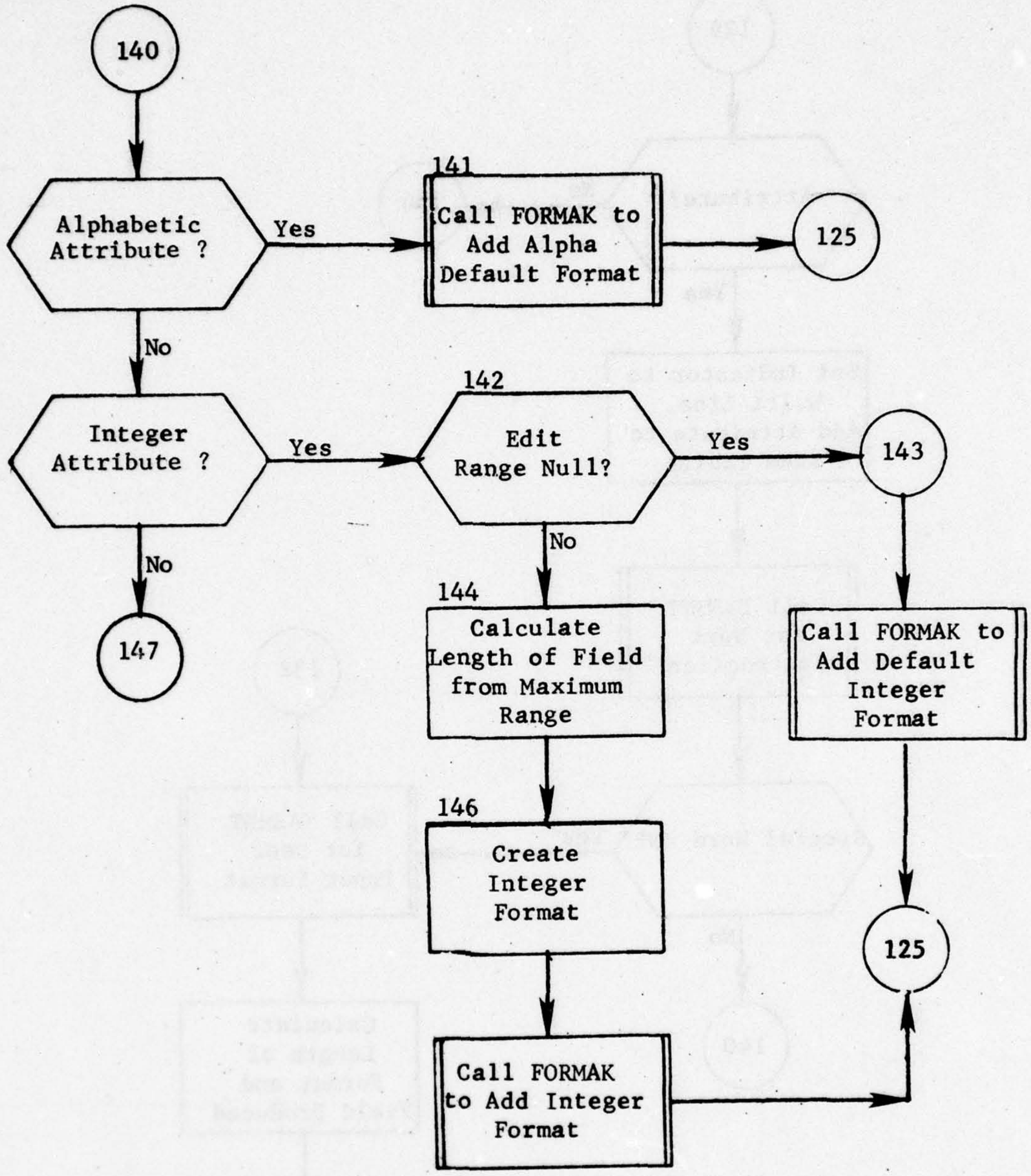


Figure 107. (Part 10 of 12)

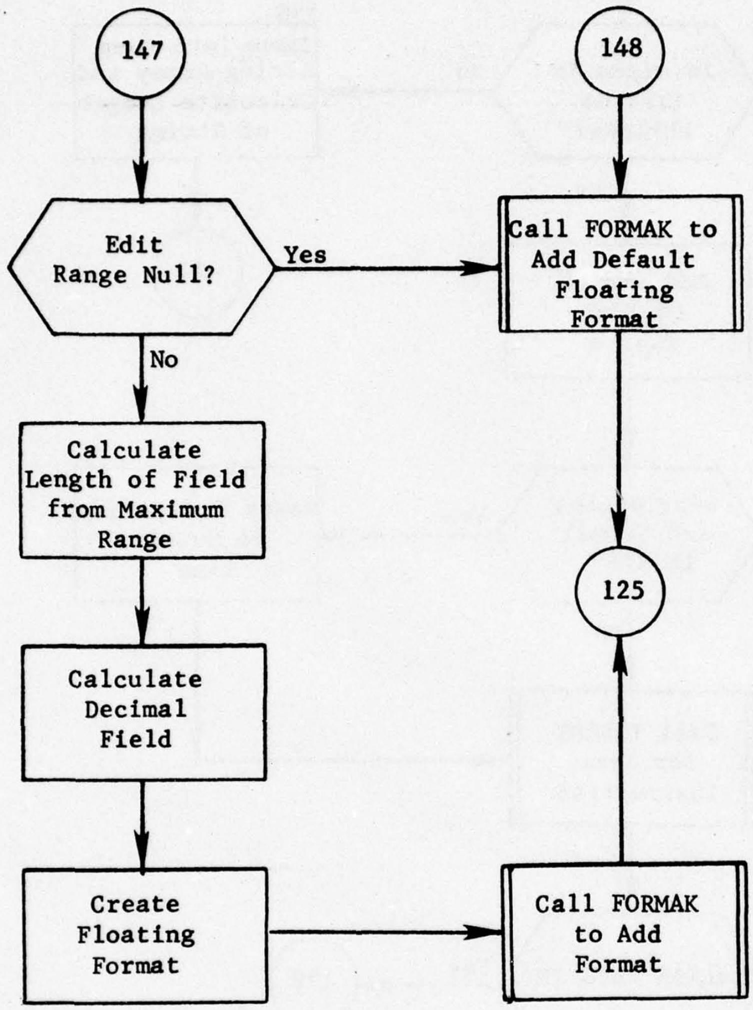


Figure 107. (Part 11 of 12)

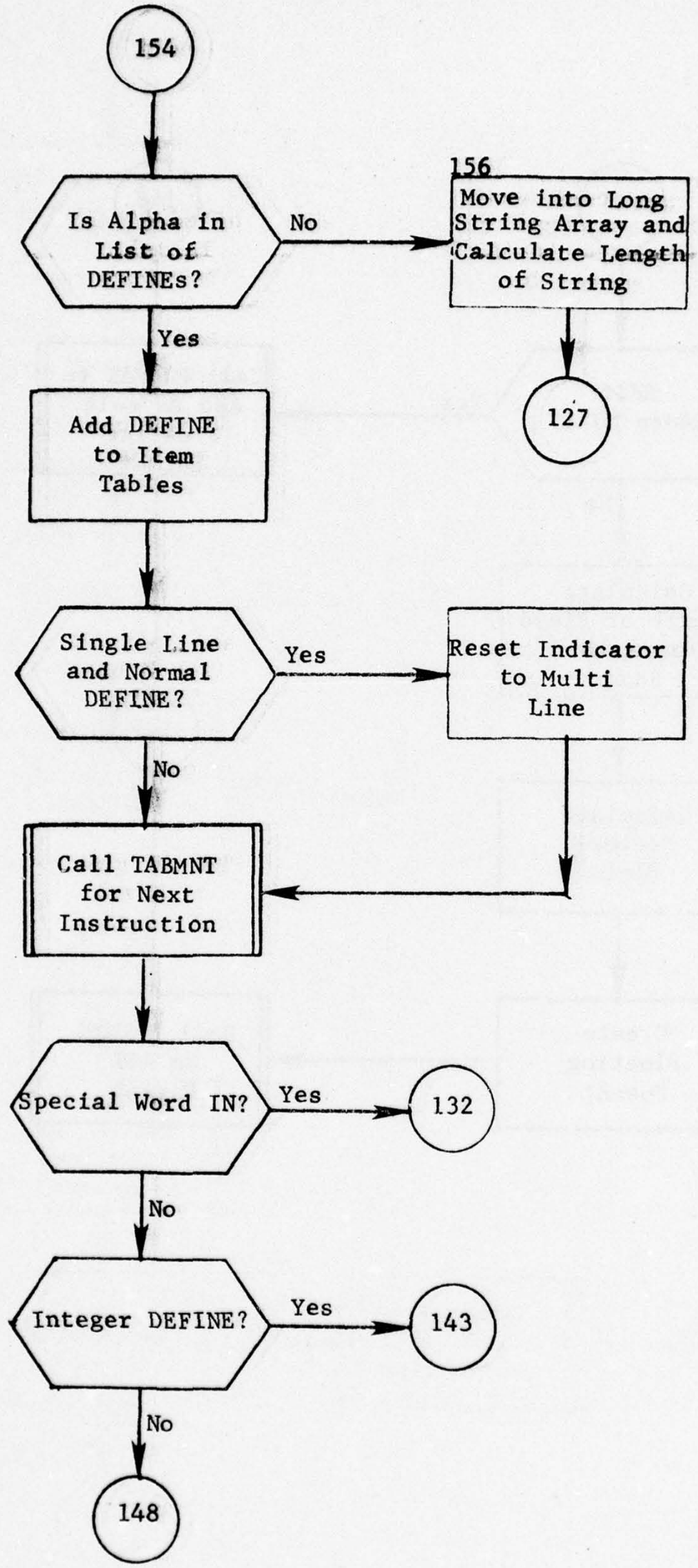


Figure 107. (Part 12 of 12)

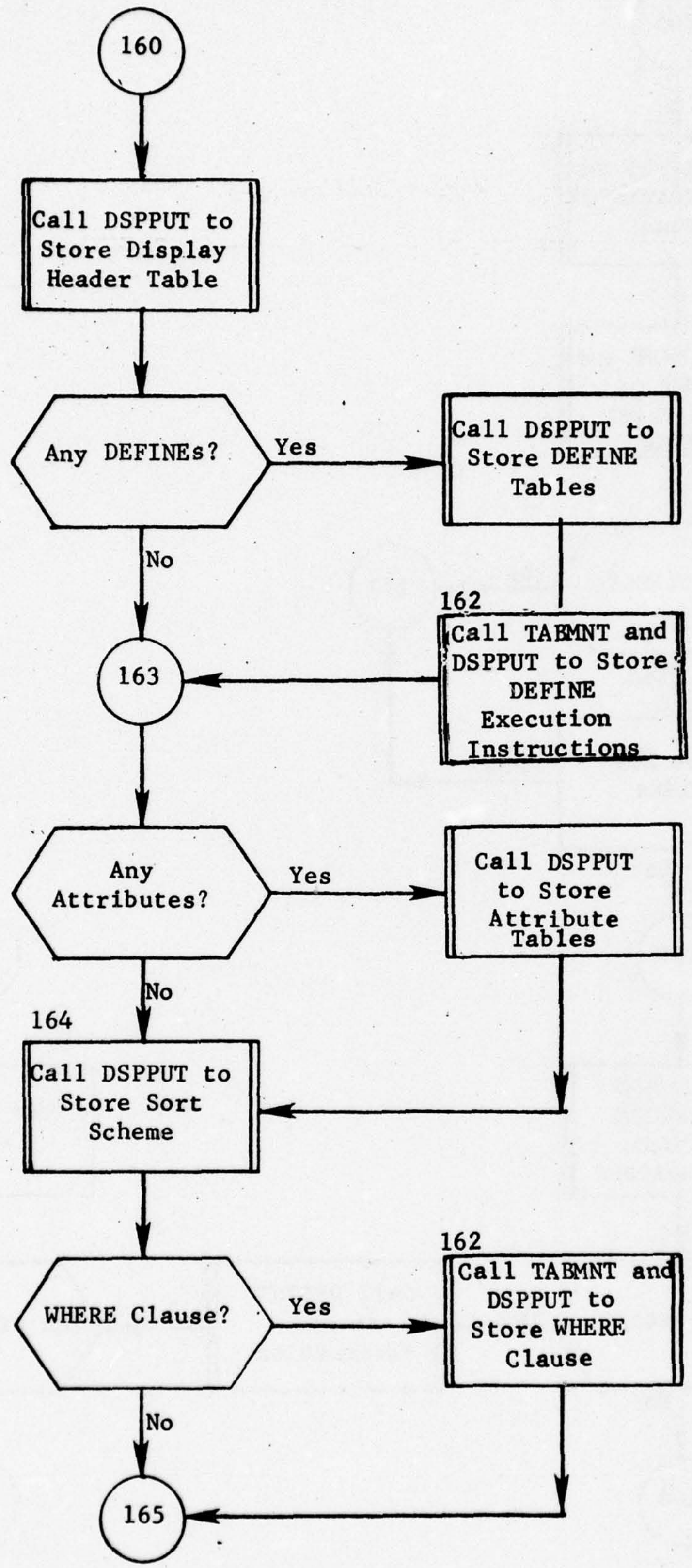


Figure 108. Subroutine DSPMAK: Step Ten (Part 1 of 4)

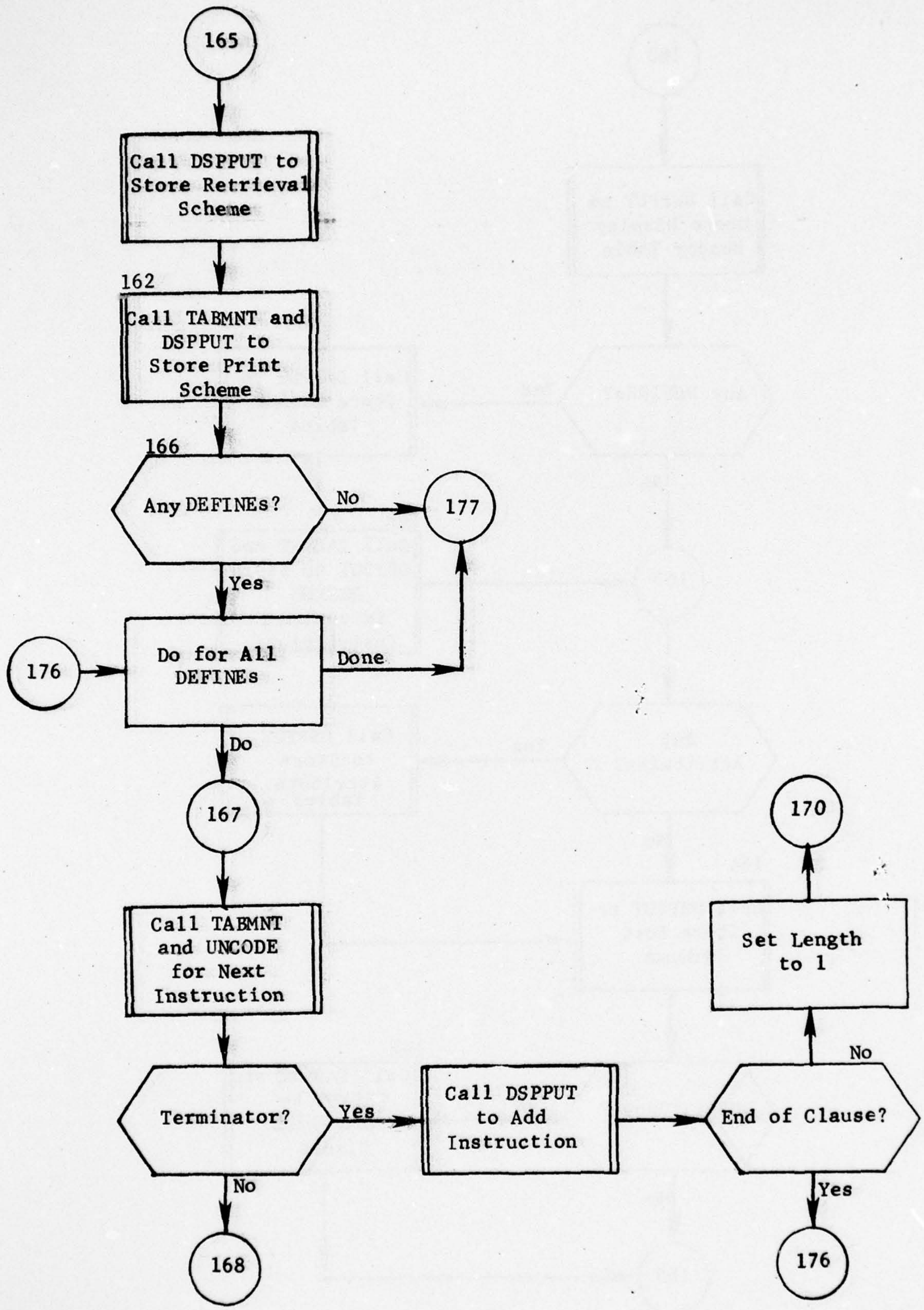


Figure 108. (Part 2 of 4)

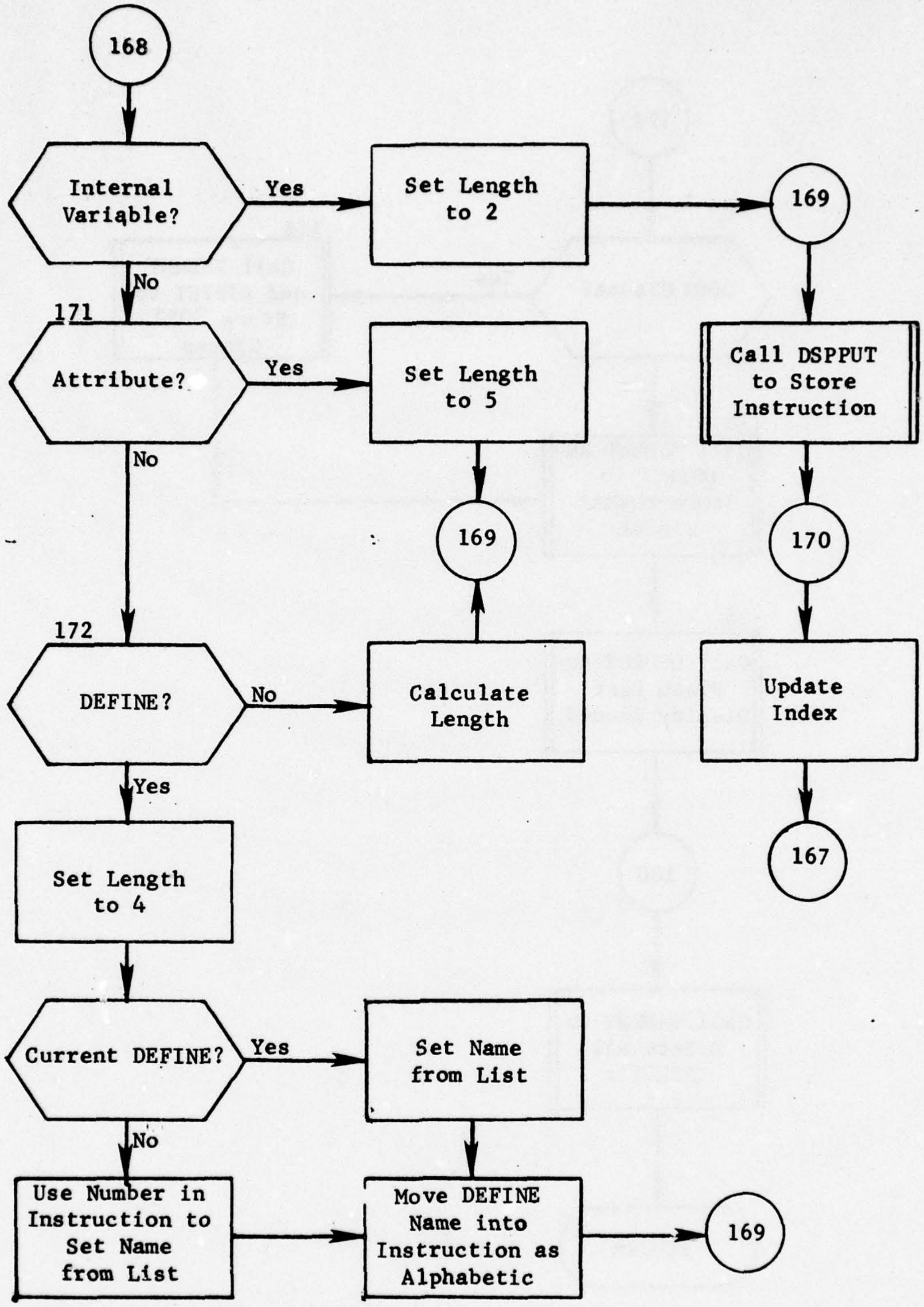


Figure 108. (Part 3 of 4)

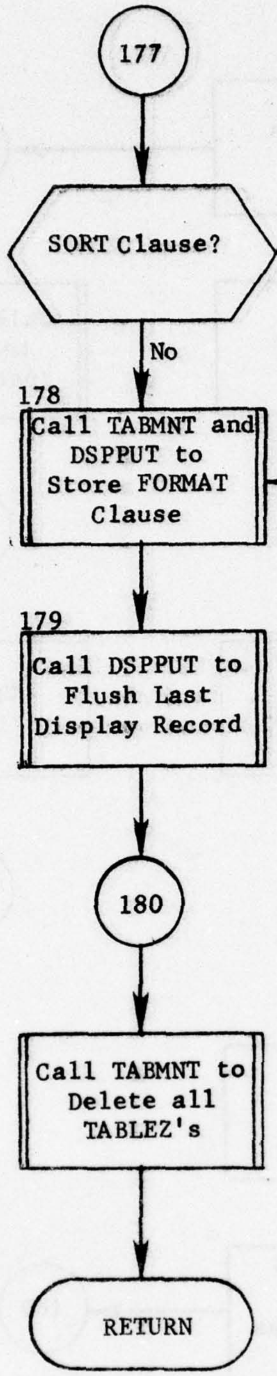


Figure 108. (Part 4 of 4)

6.11 Subroutine PRINCE*

PURPOSE: To print a display

ENTRY POINTS: PRINCE

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C15, C20, C30, DEFVAR, DSPHED, IDPT, INS, NONPAG, OOPS, PAGPAG, PSCOM, SCHEME, SORSCH, ZEES

SUBROUTINES CALLED: DSPGET, GETNXT, HDFND, HEAD, INSFLS, INSGET, INSPUT, NEXTTT, PSNXT, PSPUT, PSREC, PSRWD, RETRV, TABMNT, UNCODE, XDEFN, XSORT, XWHERE

CALLED BY: ENTMOD (REPORT)

Method:

Step One

The input is scanned for the DISPLAY clause and the display name is found (if no DISPLAY clause is provided the name 'QTEMPORARYQQ' is used). The DSPLAY chain is searched for the appropriate display table. Next, subroutine DSPGET is called to retrieve the display table header, define tables (the execution instructions are stored in utility table 1), attribute tables and sort schemes (elements 1-70 in table 18) (see figure 109).

Step Two

The input is now scanned for a WHERE clause. If none is found the old clause is retrieved by DSPGET and stored via INSPUT in the input instruction tables (this clause is stored beginning at the next full table after those already in use). If a new WHERE clause is found it is stored via TABMNT in utility table 2. In the storing process its alphabetic instructions which referenced DEFINE names are converted as per step six, section 6.10. Also, whatever values are input for the CLASS and SIDE attributes are saved. When the WHERE clause is completely processed, it is moved from utility table 2 into the input instruction tables (see figure 110).

---

*Main routine of overlay link RPTPRN

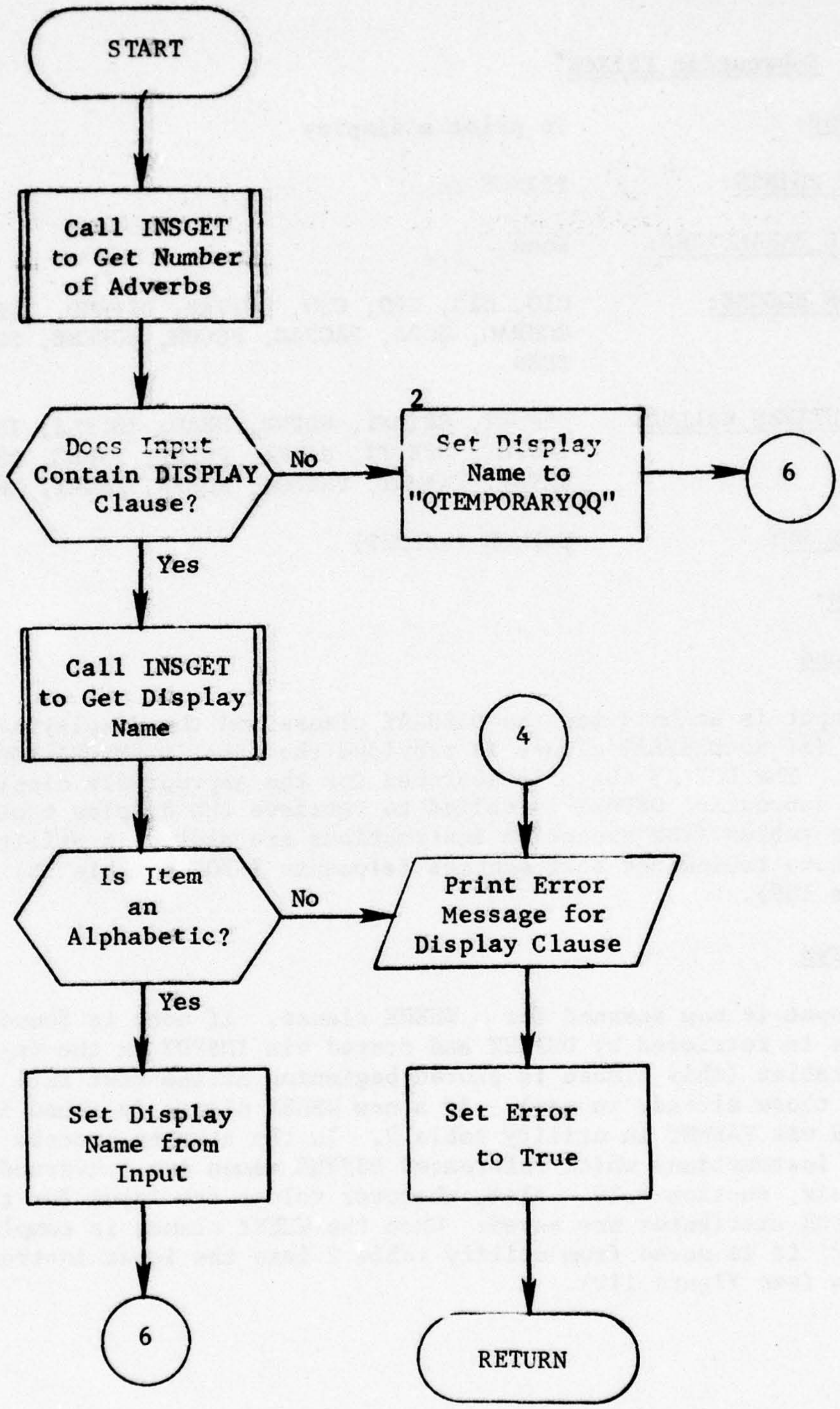


Figure 109. Subroutine PRINCE: Step One (Part 1 of 3)

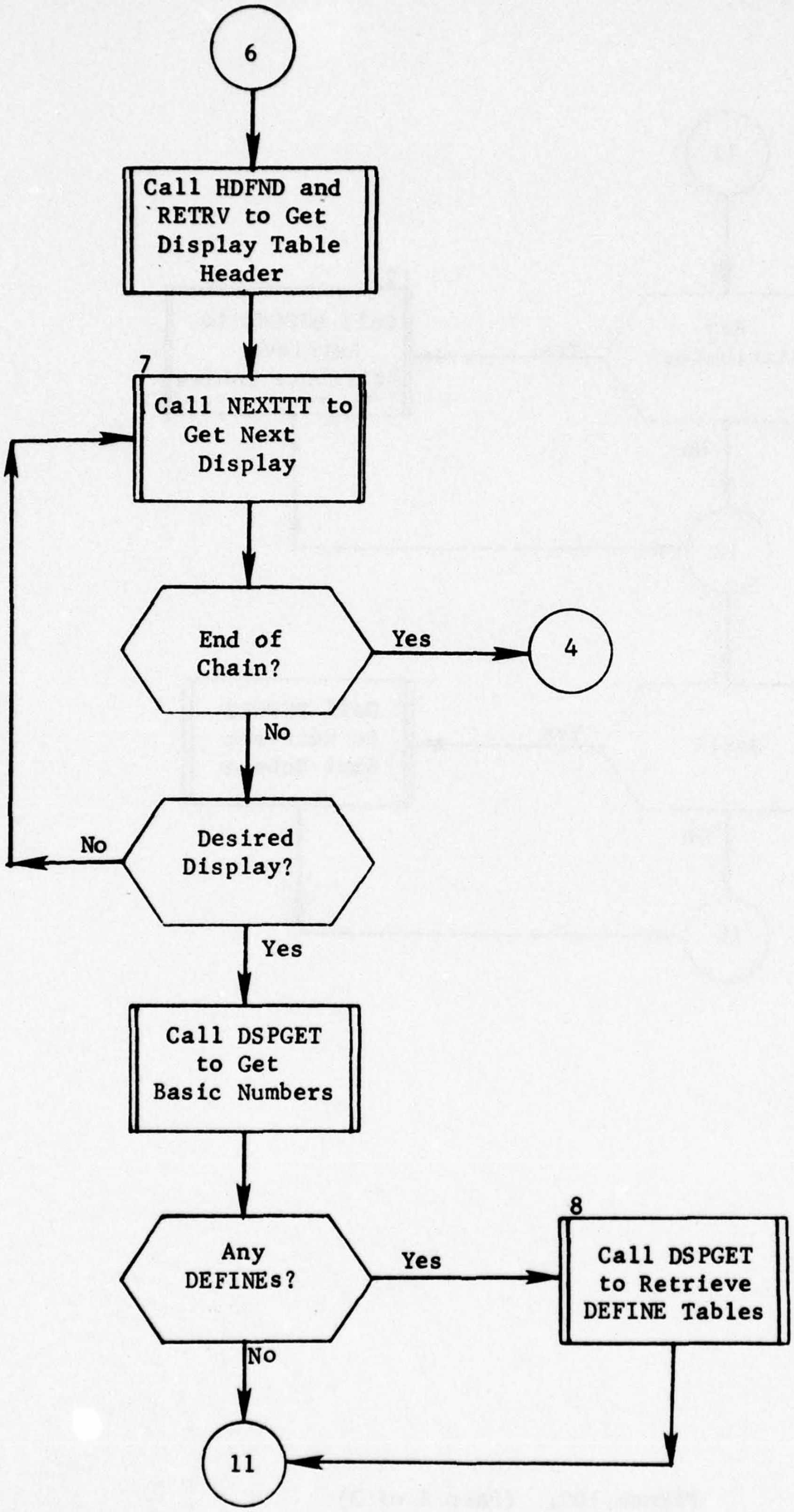


Figure 109. (Part 2 of 3)

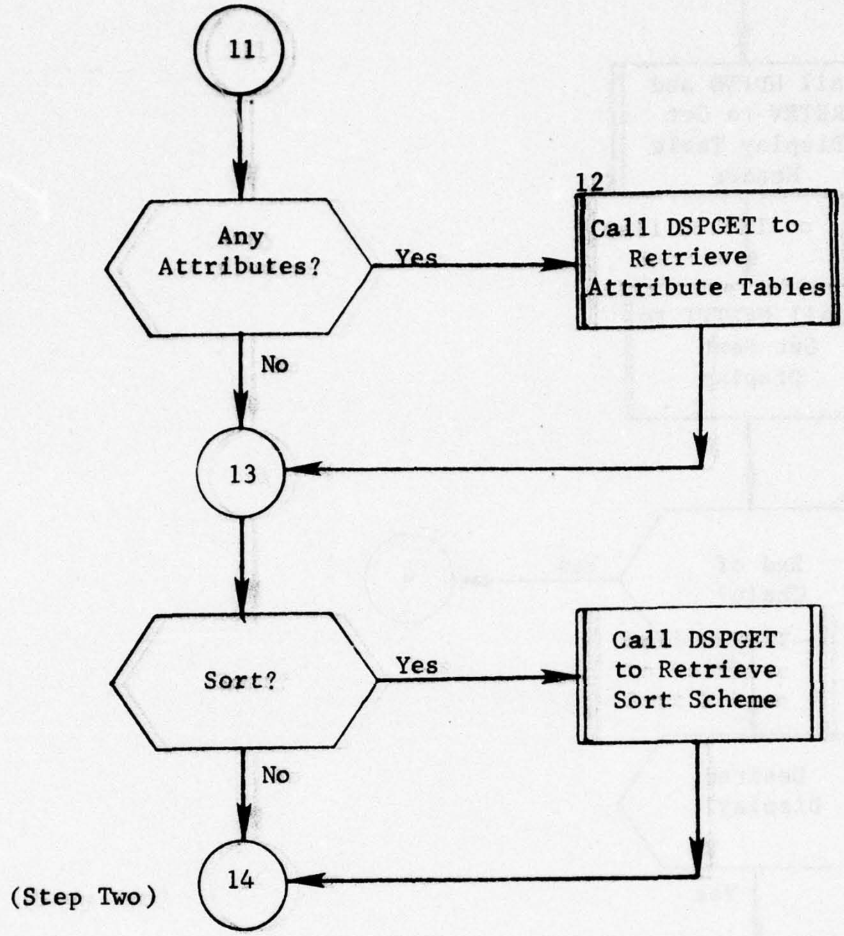


Figure 109. (Part 3 of 3)

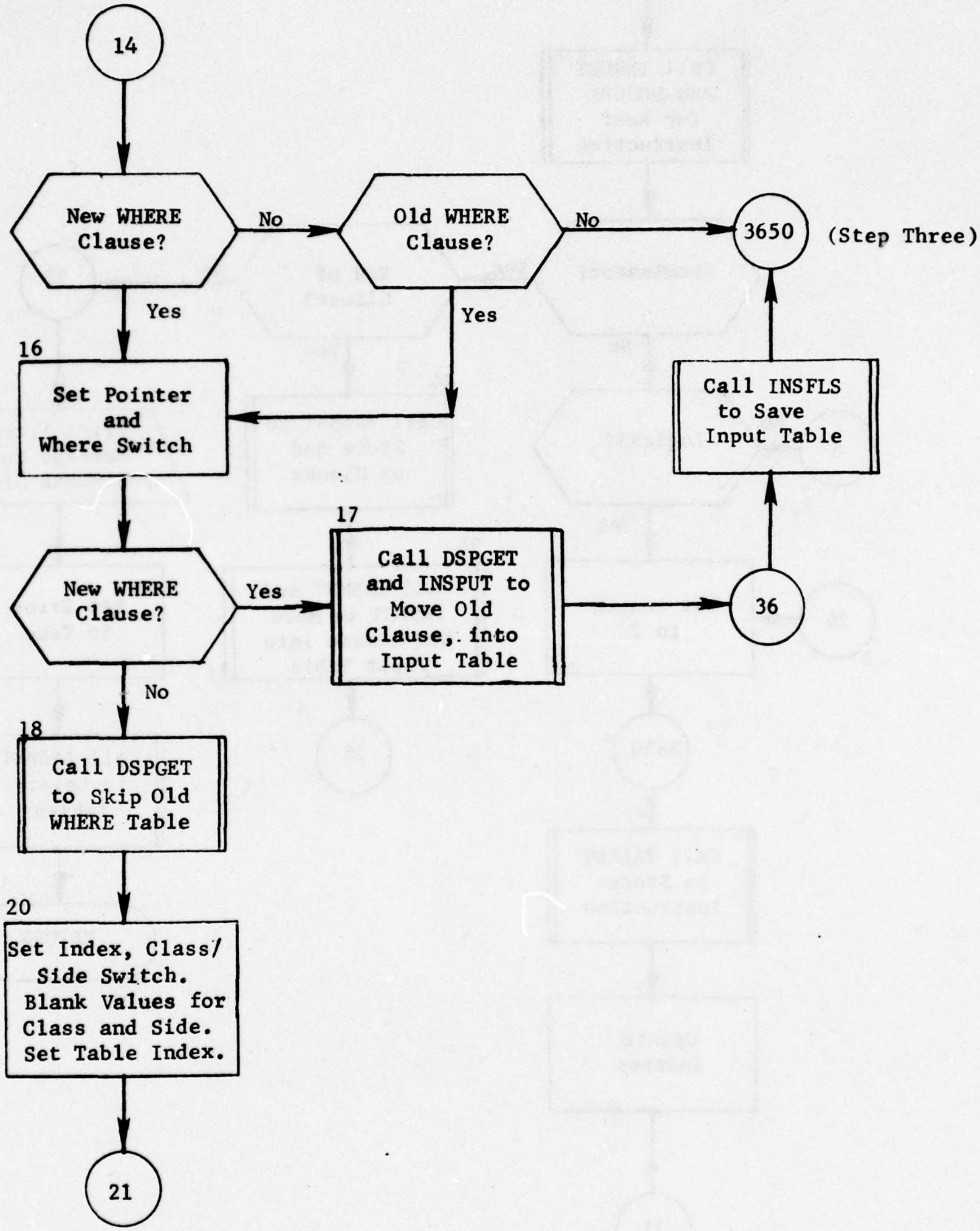


Figure 110. Subroutine PRINCE: Step Two (Part 1 of 4)

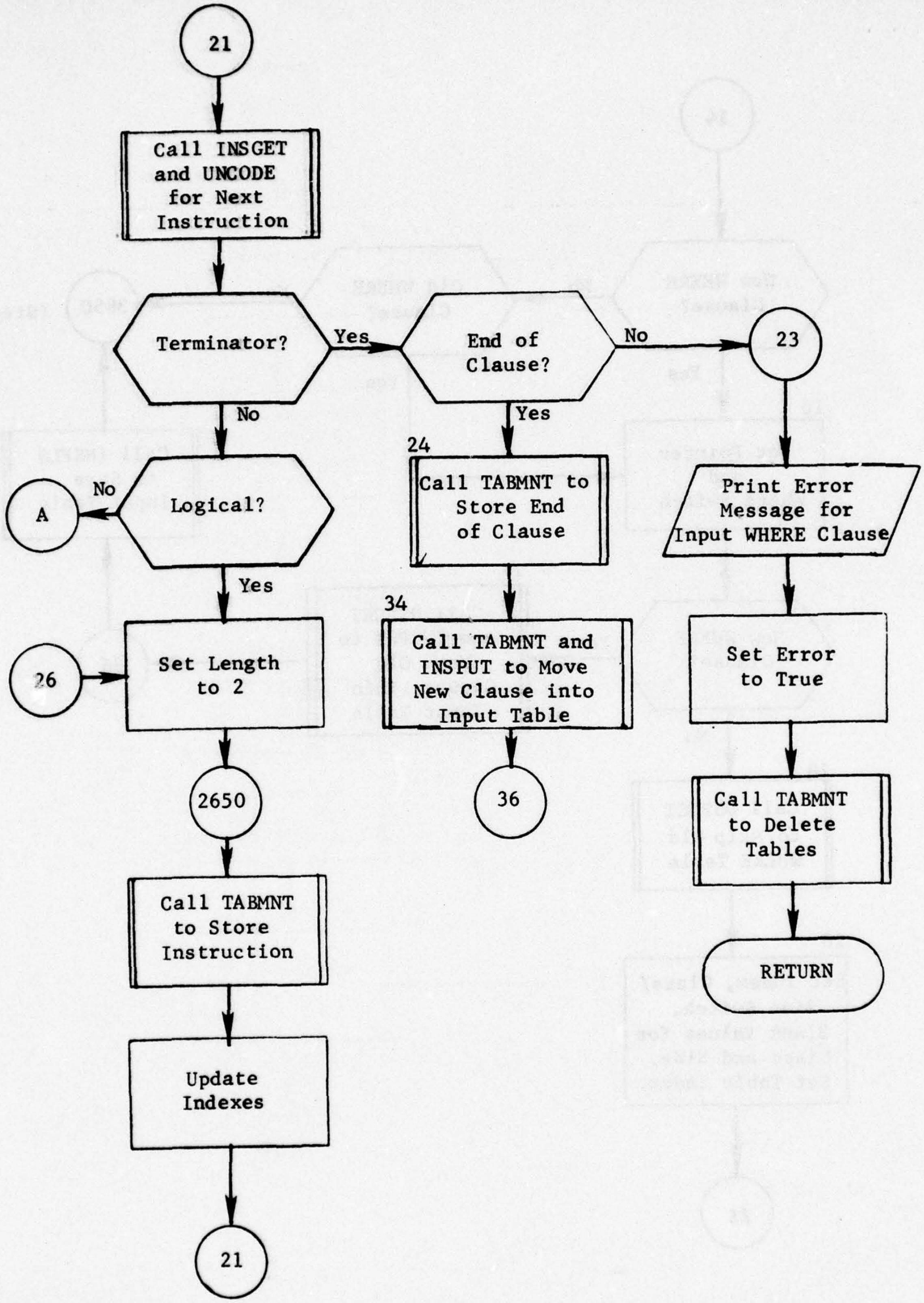


Figure 110. (Part 2 of 4)

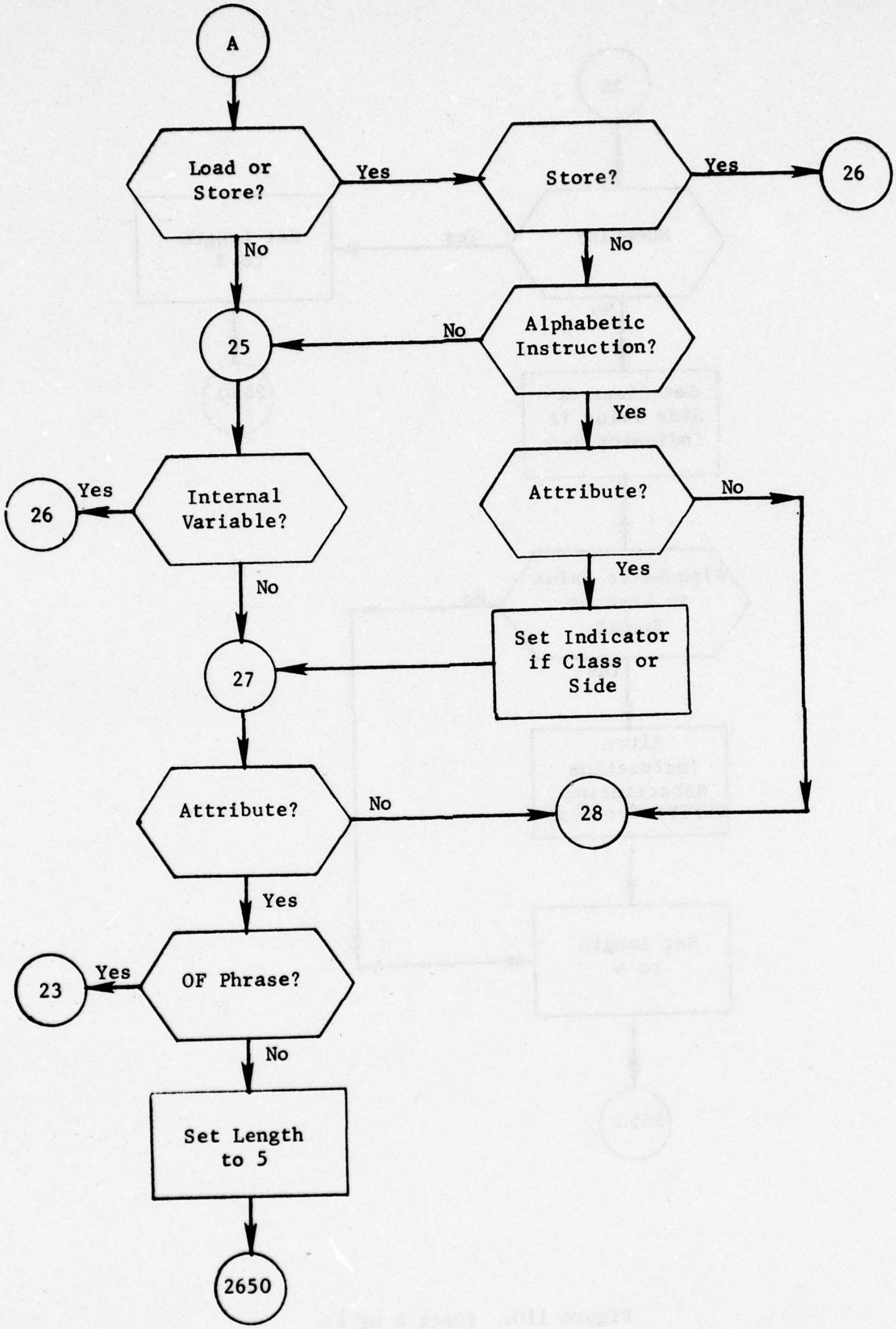


Figure 110. (Part 3 of 4)

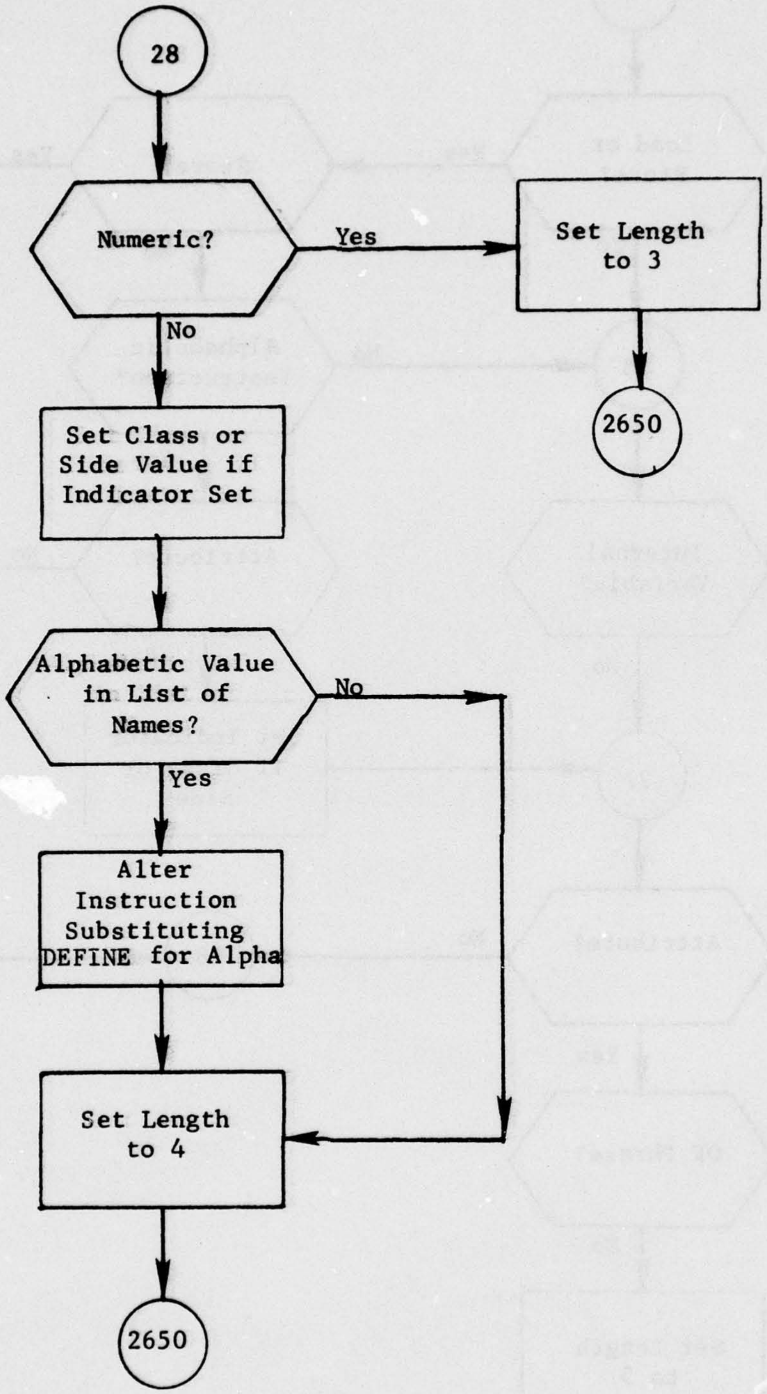


Figure 110. (Part 4 of 4)

Step Three

The retrieval scheme is now brought in by subroutine DSPGET. If the first instruction needs to be altered because of a change in the values for CLASS and SIDE in a new WHERE clause, this function is performed. This change may require the adjustment of all scheme pointers (see figure 111).

Step Four

The print/sort file is now created. The first part of the process is to call PSREC and initialize all sums and product DEFINEs. Next GETNXT is used to execute the retrieval scheme. For each return from GETNXT, XDEFN is called to execute all DEFINEs and XWHERE is called to perform selection. If a record is selected, the appropriate attributes and DEFINEs are stored in the RECORD array and PSPUT is called to add a new print/sort record. Finally when the retrieval scheme is complete, XSORT is called to sort the print/sort file (see figure 112).

Step Five

The print scheme is now retrieved through subroutine DSPGET as needed and executed. The process is performed separately for each PAGE. For each PAGE all HEADER items are retrieved and stored in utility table 3. Next, all TRAILER items are retrieved and stored in utility table 4. Now the set of HEADER items is executed and their corresponding print produced. Then each LINE item is retrieved and its corresponding print produced. If the LINE is a single logical line, this is a simple execution of the item. For multiple LINES, the process begins with the call of PSRWD and the initialization of sums and products. Then PSNXT is called for each print/sort record. When it is retrieved, XDEFN is called for all DEFINEs that are not normal. Finally, the print line is produced and PSNXT is called again.

During the LINE print process, PRINCE keeps track of the number of lines remaining on the page. When there is only room left for the TRAILERS, they are executed and the HEADERS are executed and processing continues. When all LINE items have been processed the TRAILERs are executed.

The above process takes place for each PAGE item (see figure 113).

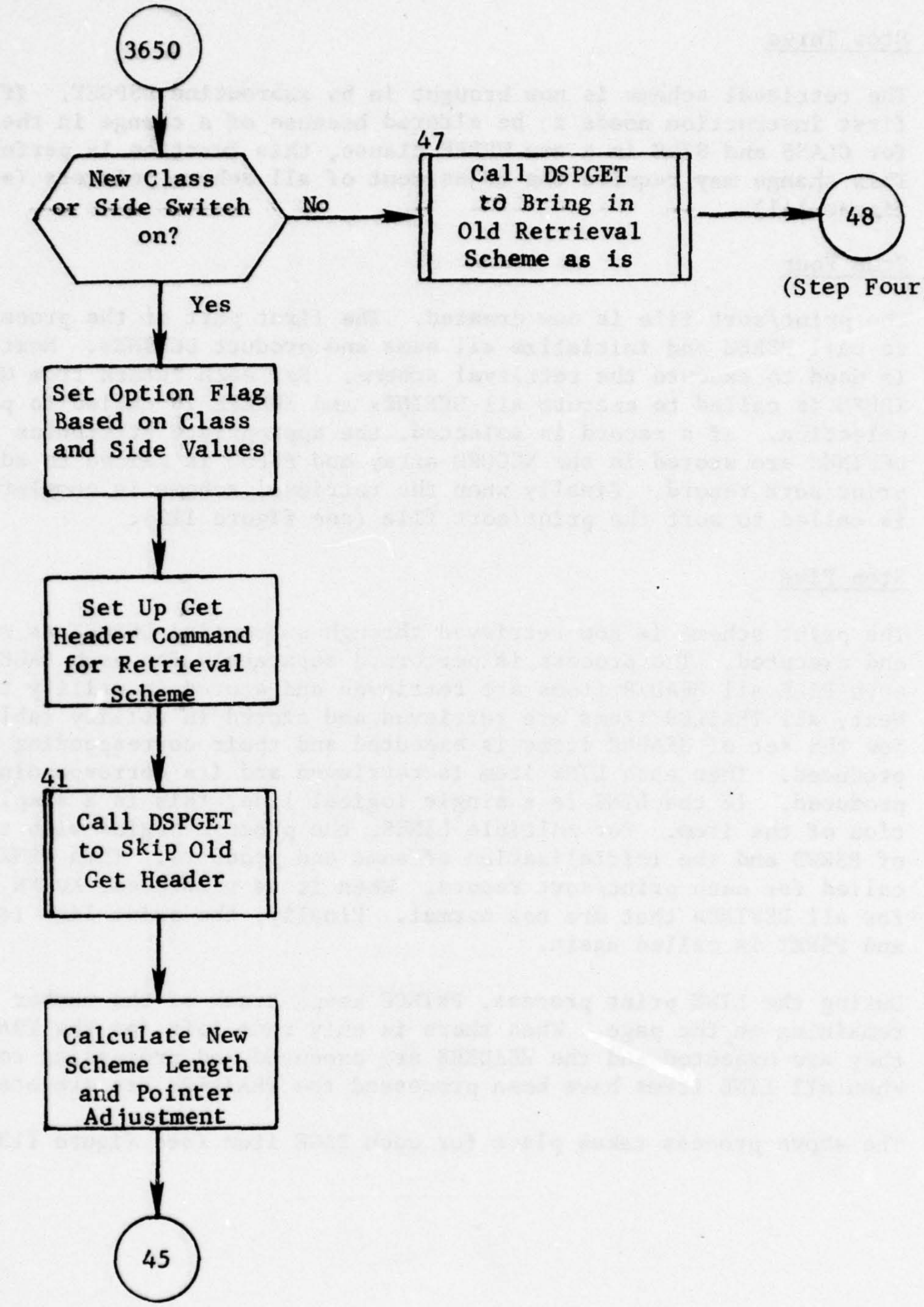


Figure 111. Subroutine PRINCE: Step Three (Part 1 of 2)

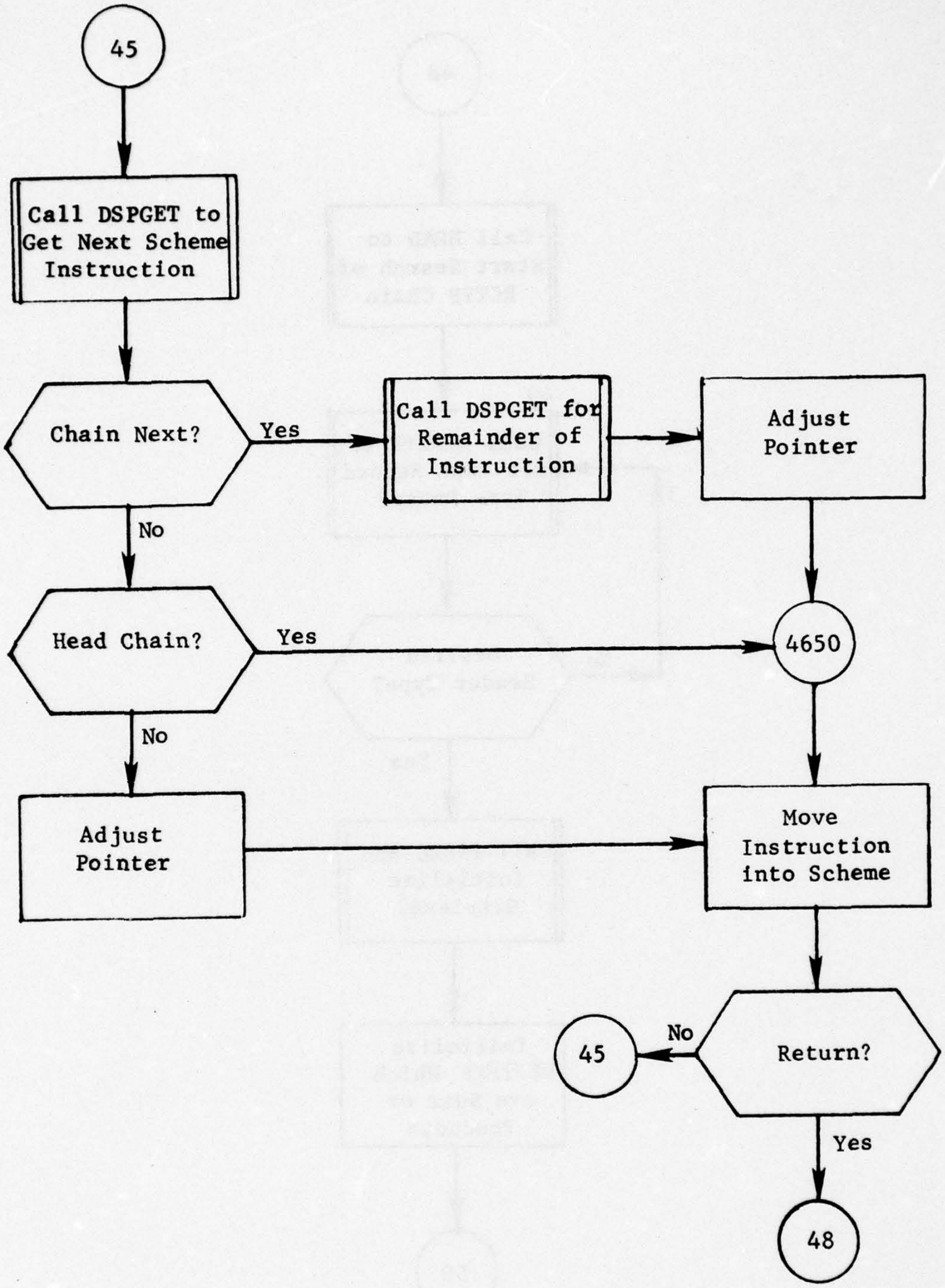


Figure 111. (Part 2 of 2)

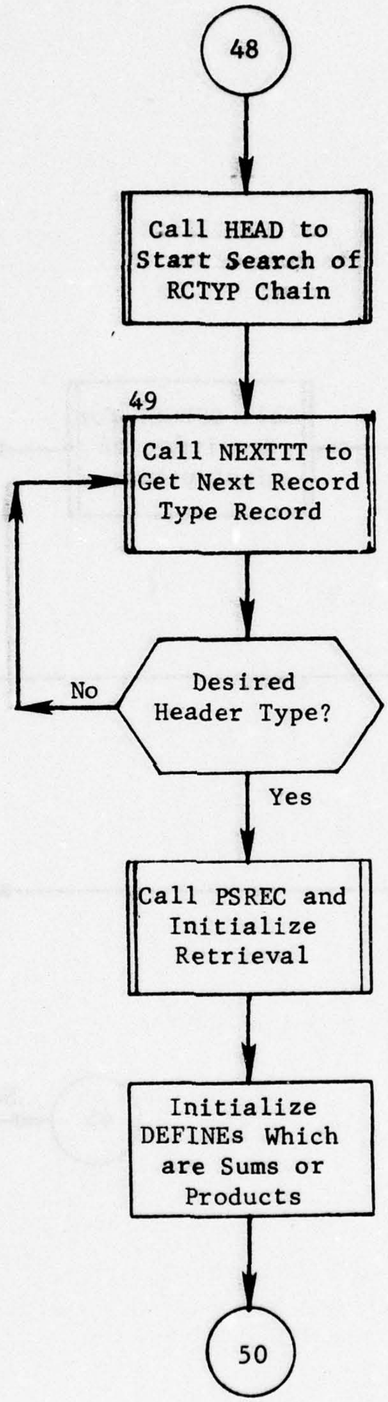


Figure 112. Subroutine PRINCE: Step Four (Part 1 of 2)

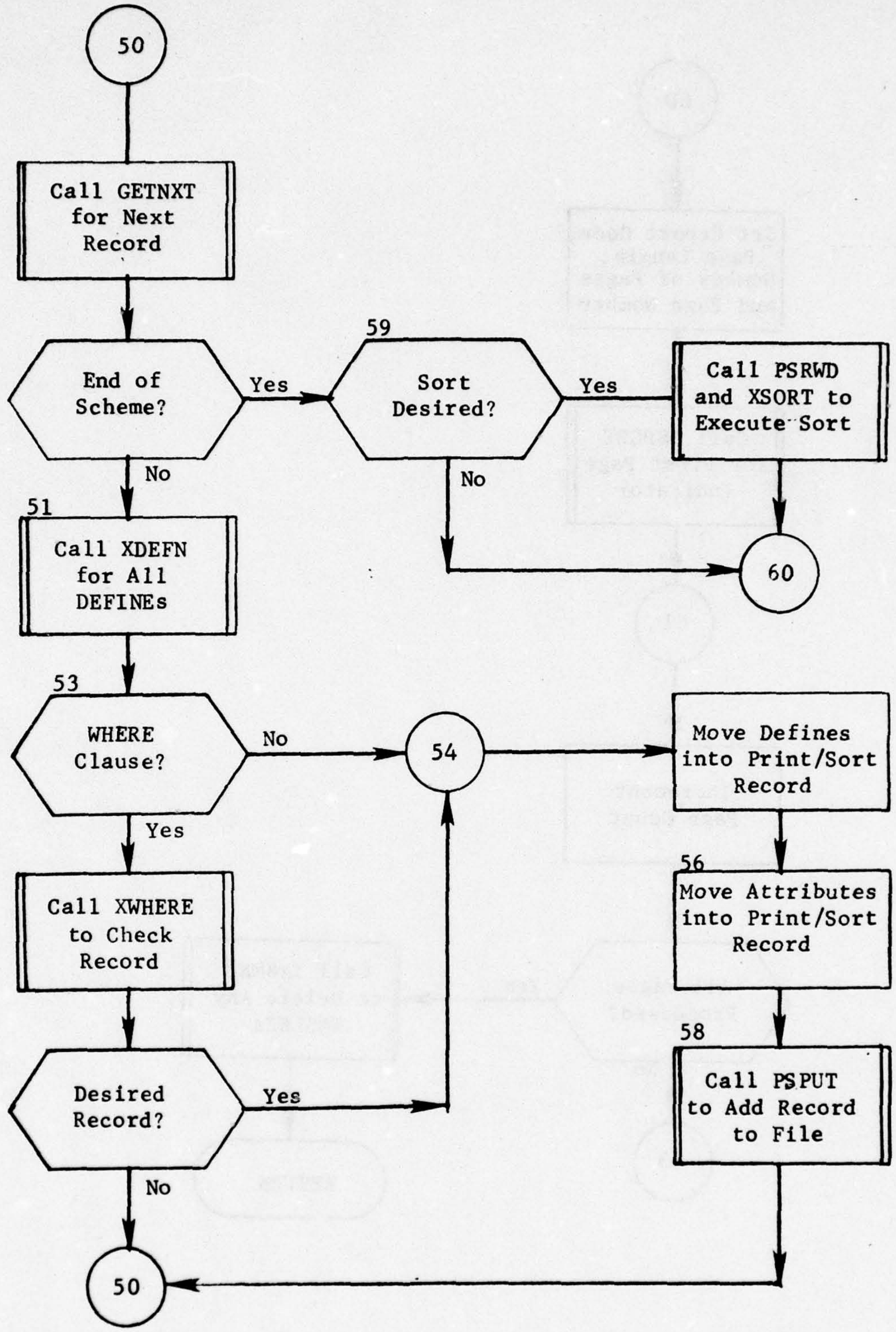


Figure 112. (Part 2 of 2)

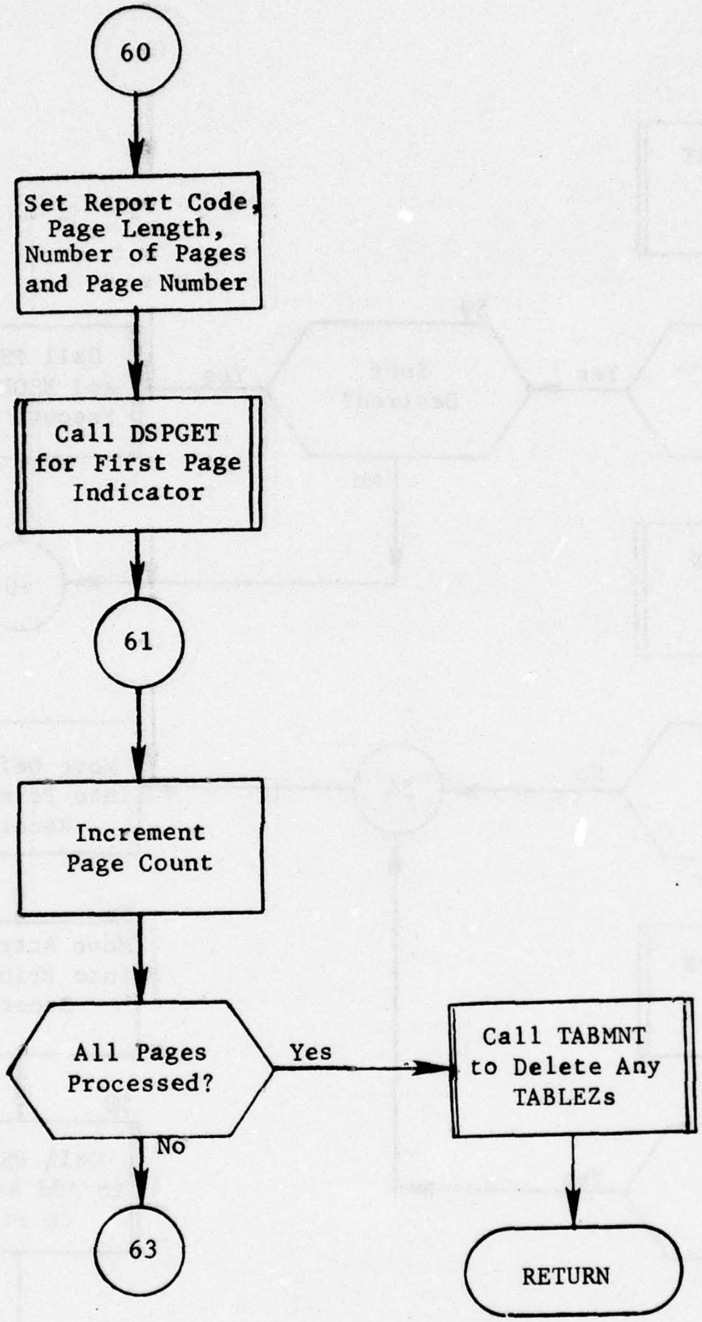


Figure 113. Subroutine PRINCE: Step Five (Part 1 of 8)

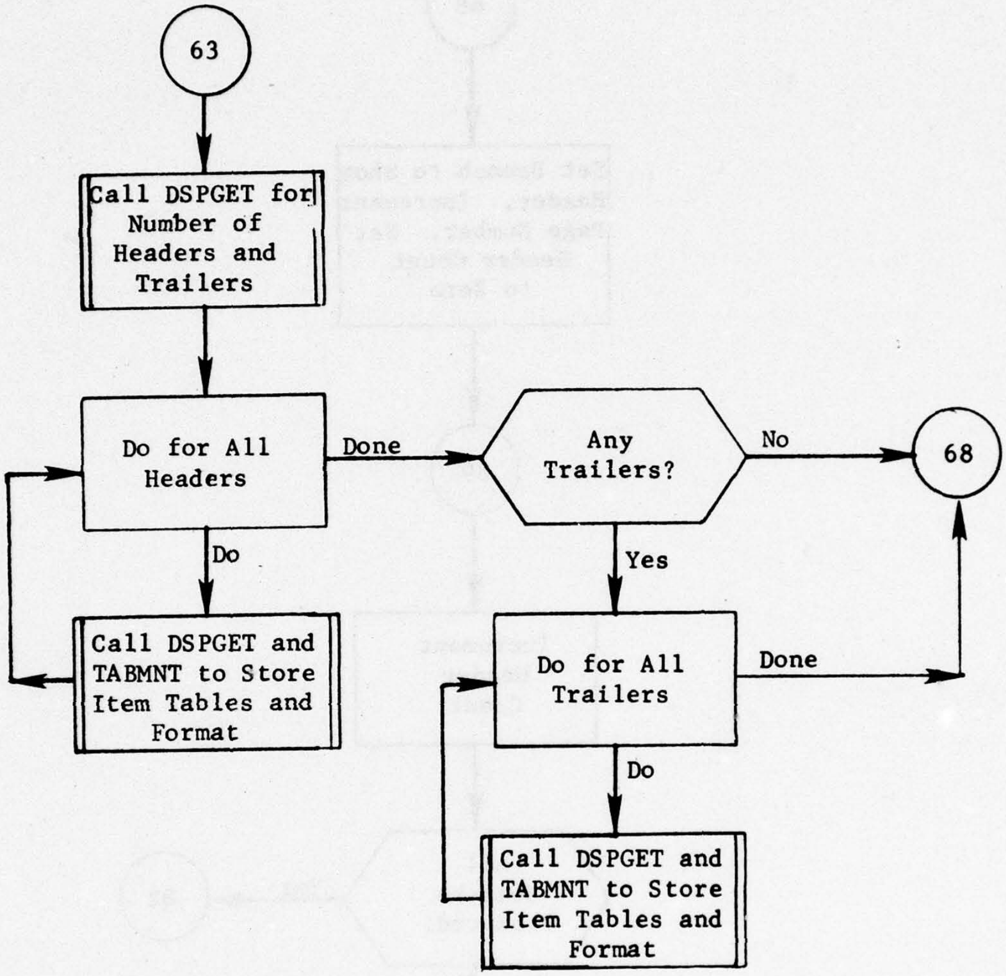


Figure 113. (Part 2 of 8)

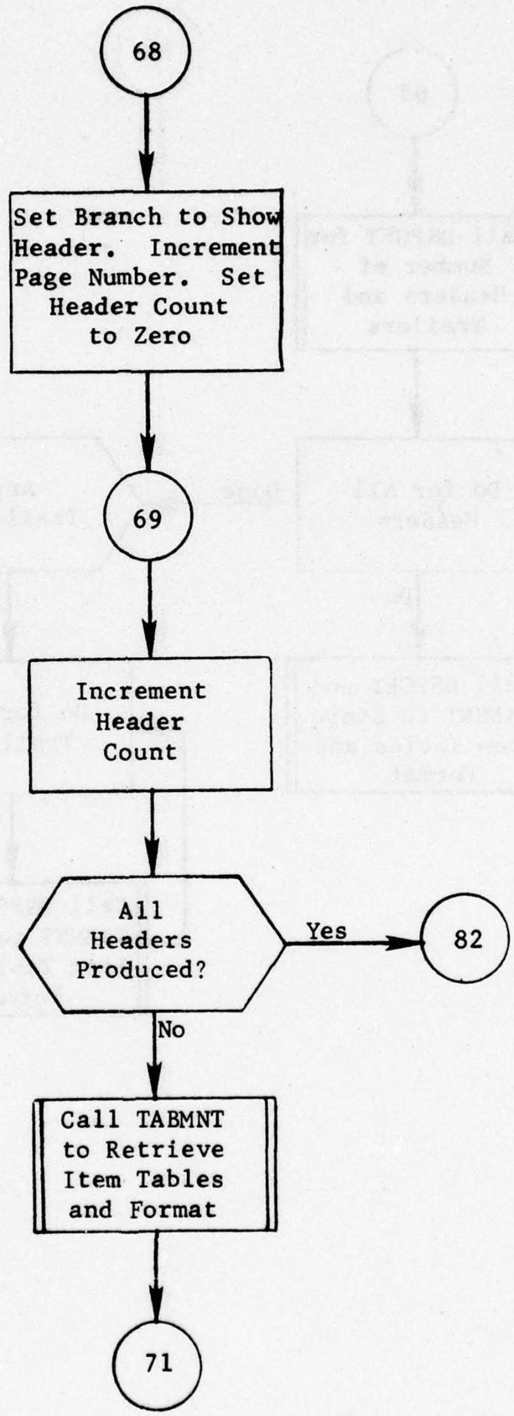


Figure 113. (Part 3 of 8)

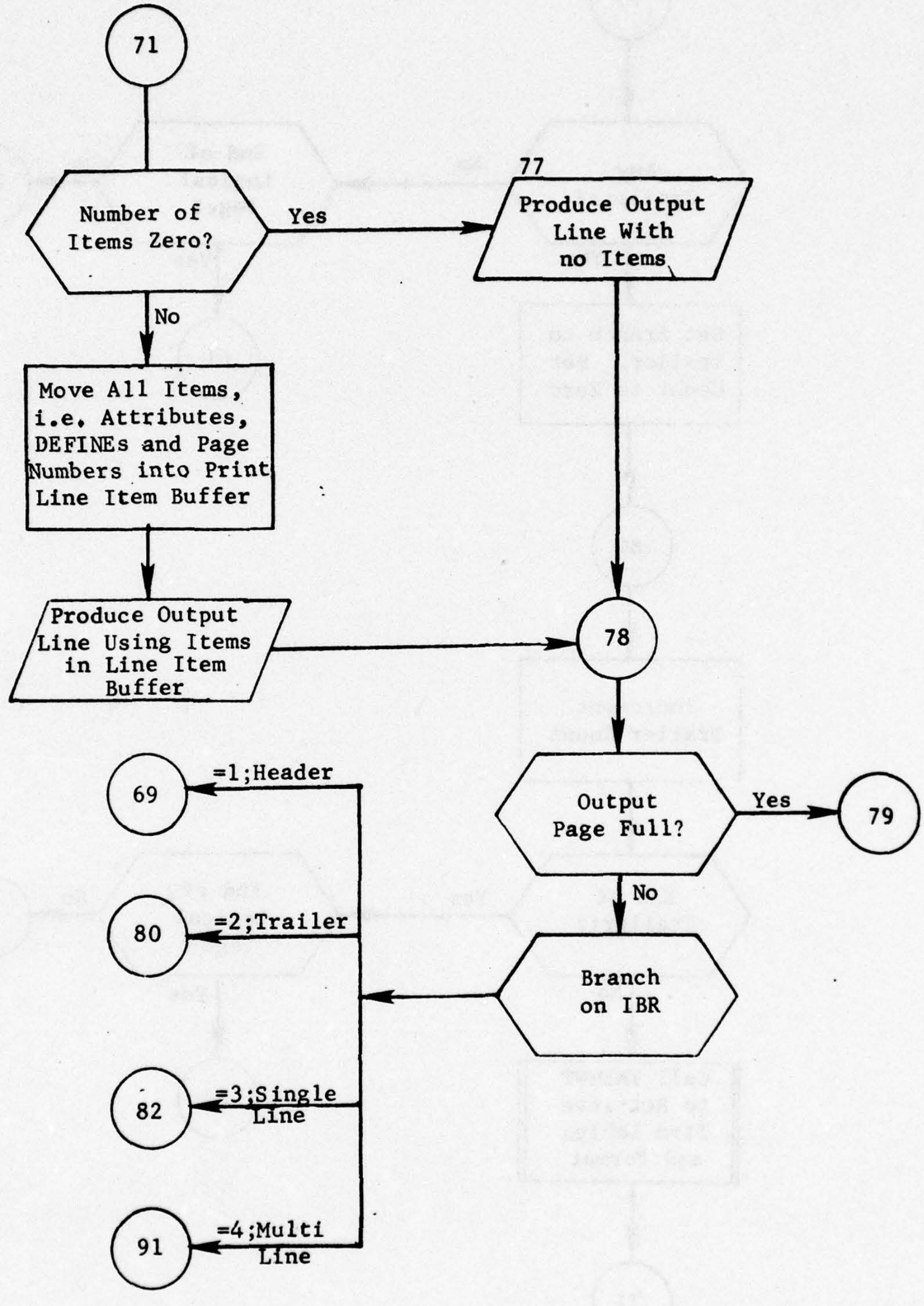


Figure 113. (Part 4 of 8)

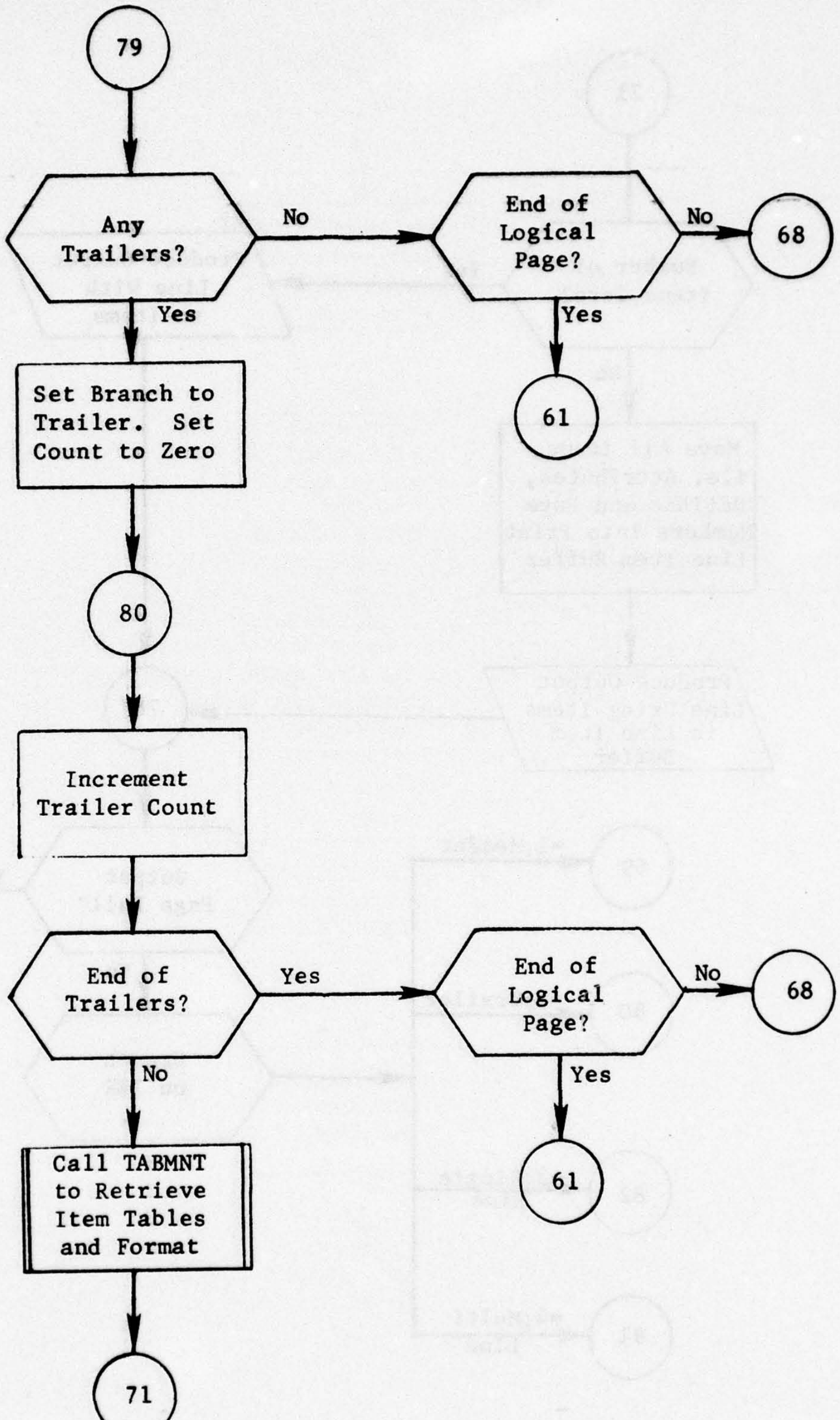


Figure 113. (Part 5 of 8)

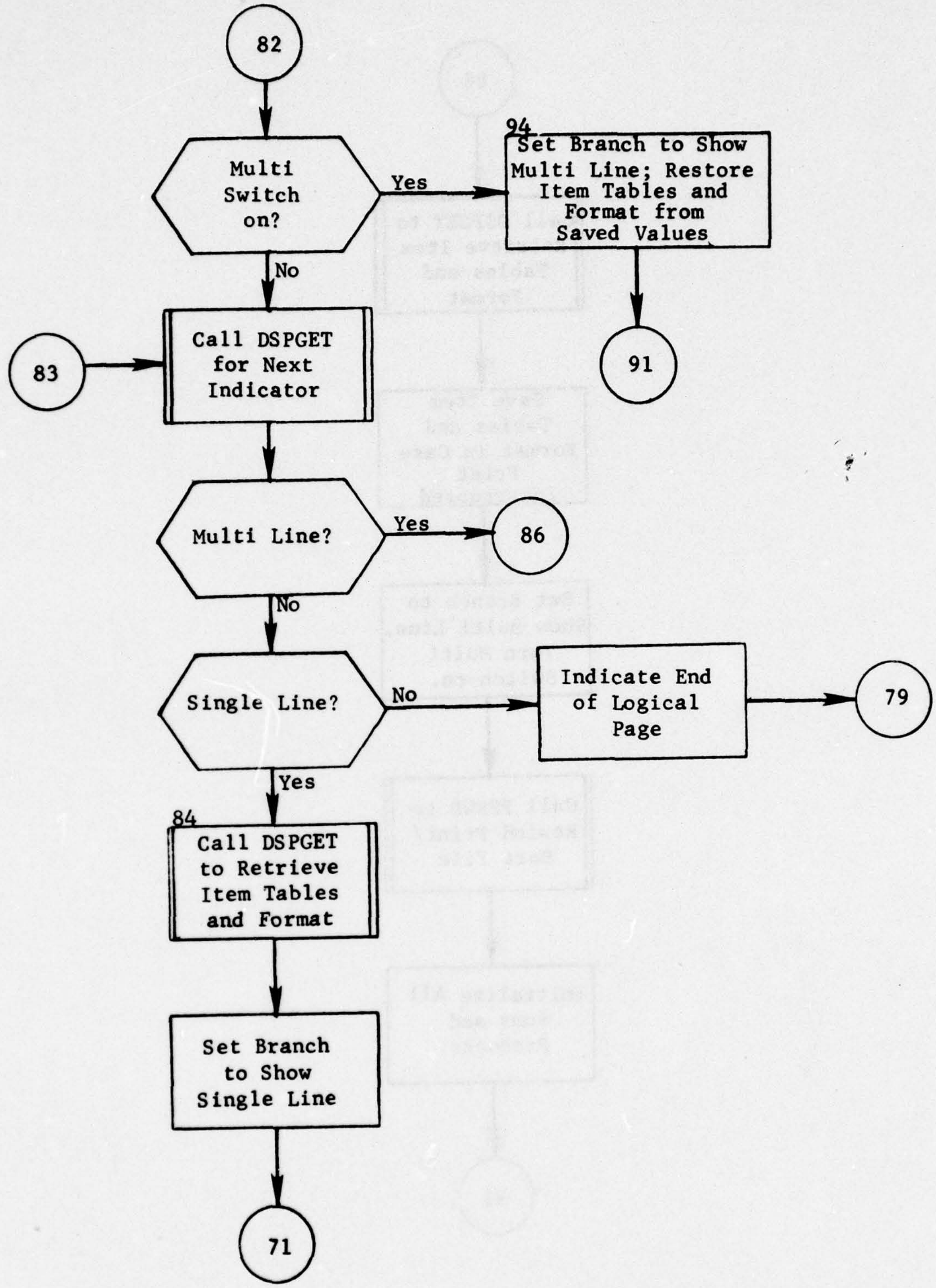


Figure 113. (Part 6 of 8)

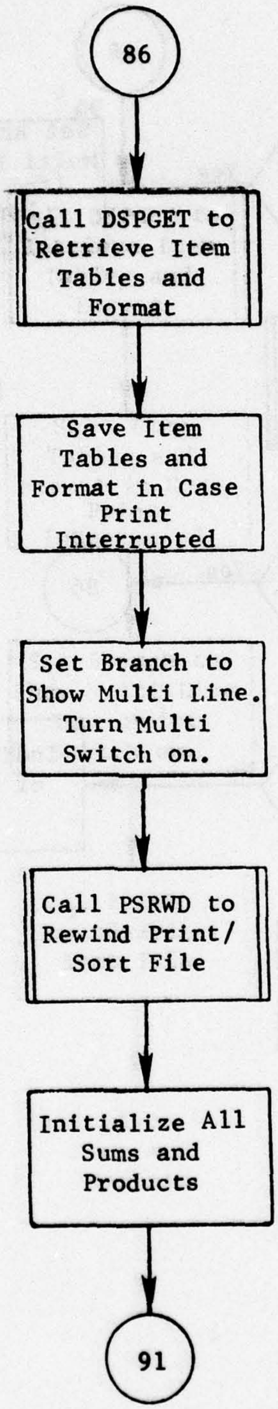


Figure 113. (Part 7 of 8)

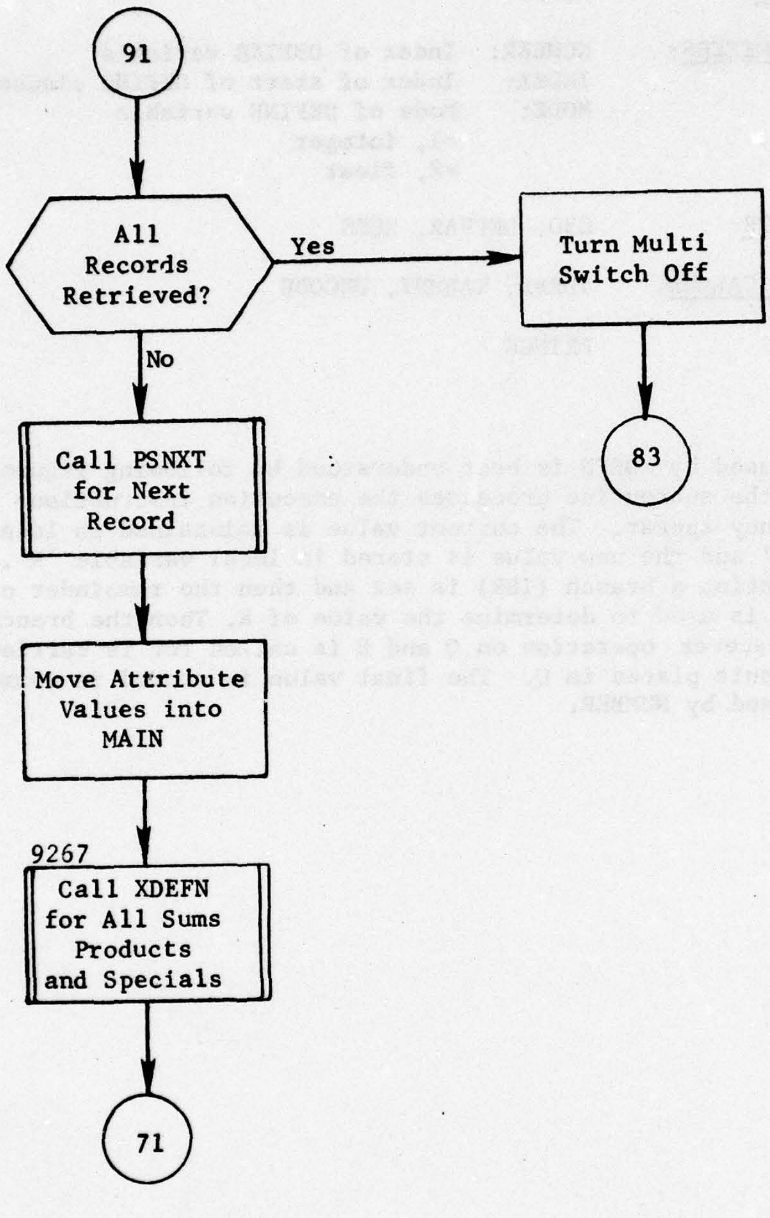


Figure 113. (Part 8 of 8)

6.11.1 Subroutine XDEFN

PURPOSE:	To execute a DEFINE clause	
ENTRY POINTS:	XDEFN	
FORMAL PARAMETERS:	NUMBER:	Index of DEFINE variable
	INDEX:	Index of start of DEFINE clause in table
	MODE:	Mode of DEFINE variable =1, integer =2, float
COMMON BLOCKS:	C30, DEFVAR, ZEES	
SUBROUTINES CALLED:	IORFL, TABMNT, UNCODE	
CALLED BY:	PRINCE	

Method:

The method used by XDEFN is best understood by following figure 114. Basically, the subroutine processes the execution instructions for a DEFINE as they appear. The current value is maintained in local variable 'Q' and the new value is stored in local variable 'R'. For each instruction a branch (IBR) is set and then the remainder of the instruction is used to determine the value of R. Then the branch is made and whatever operation on Q and R is called for is carried out with the result placed in Q. The final value is stored in common block DEFVAR indexed by NUMBER.

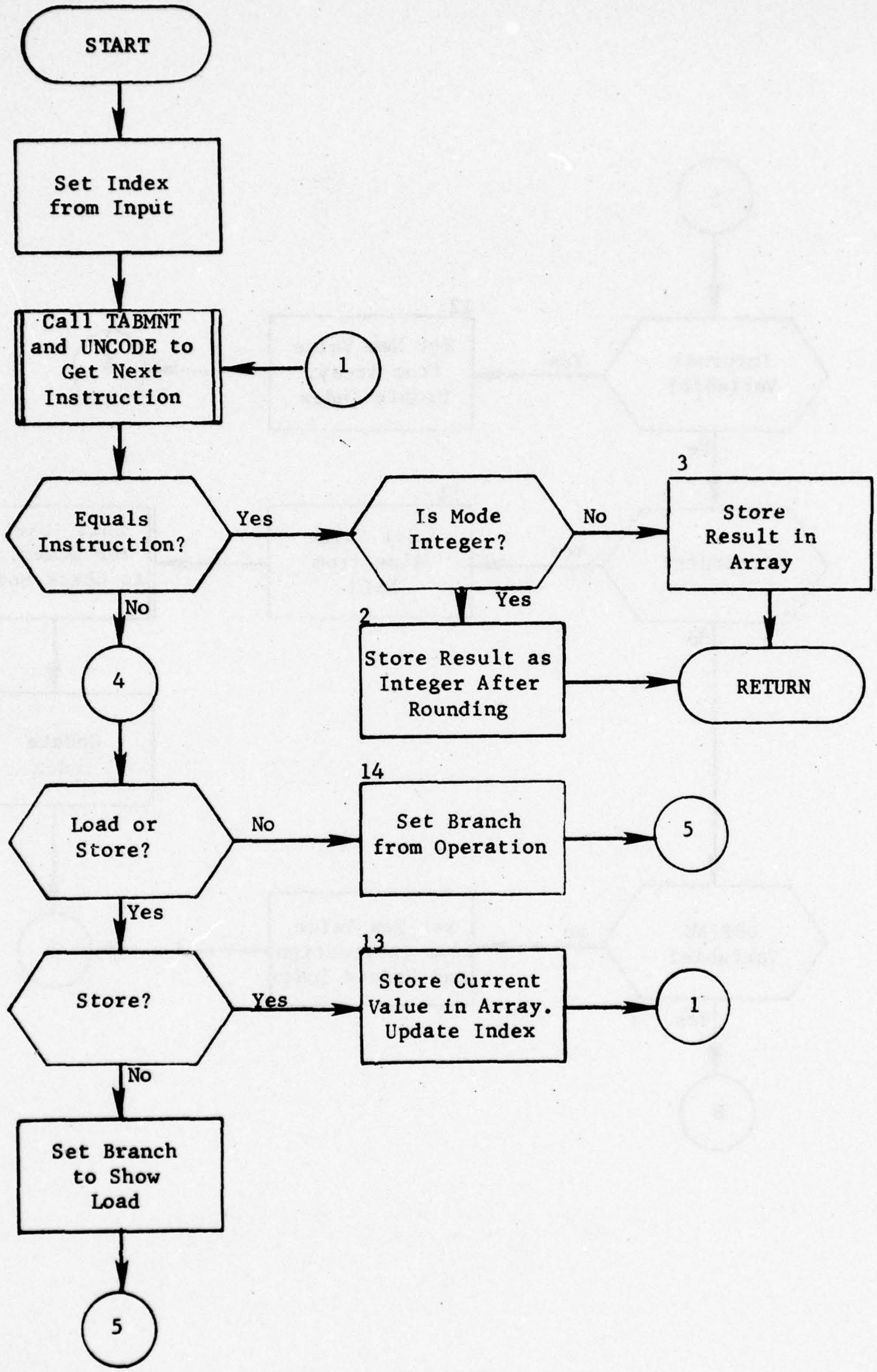


Figure 114. Subroutine XDEFN (Part 1 of 4)

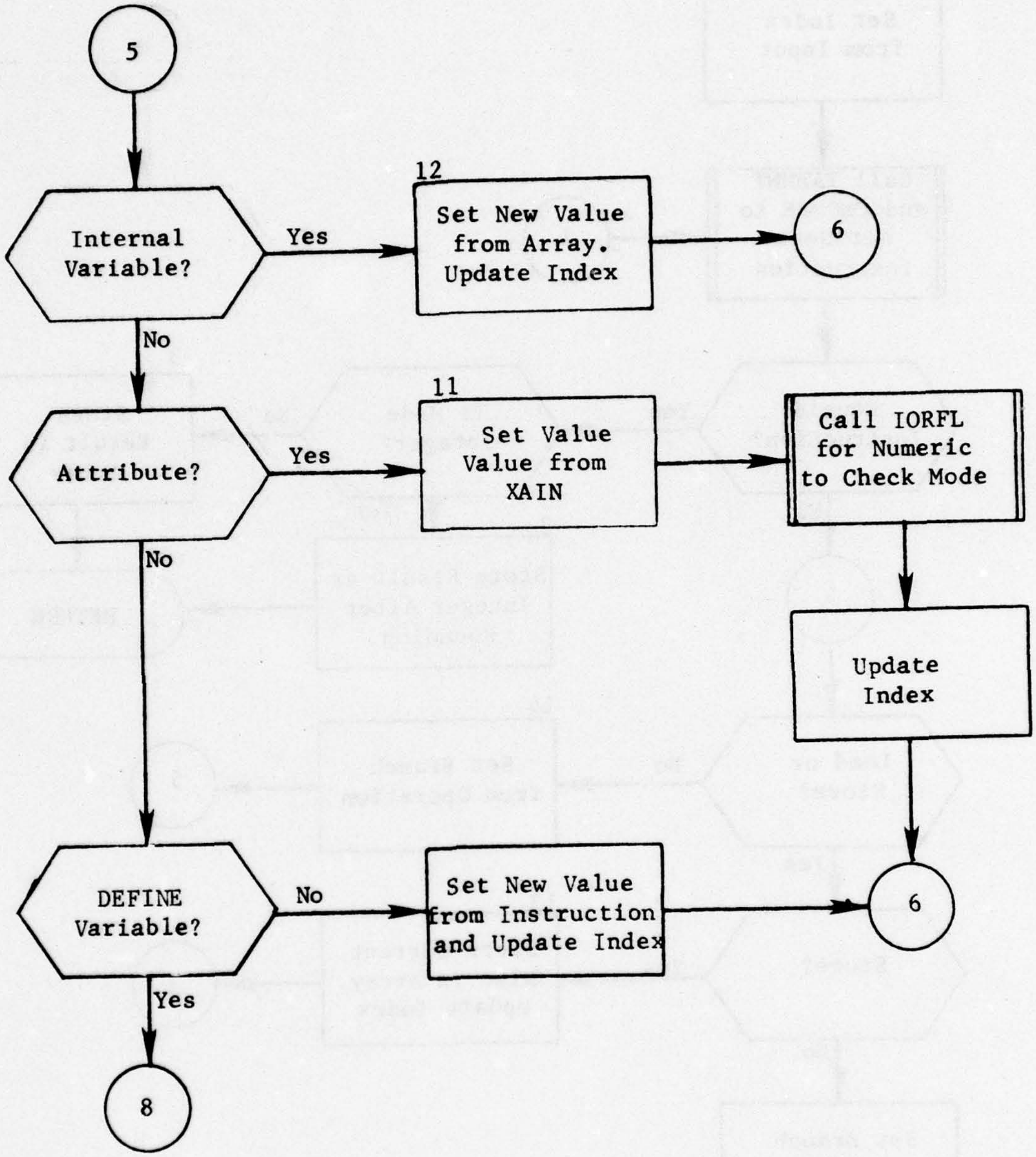


Figure 114. (Part 2 of 4)

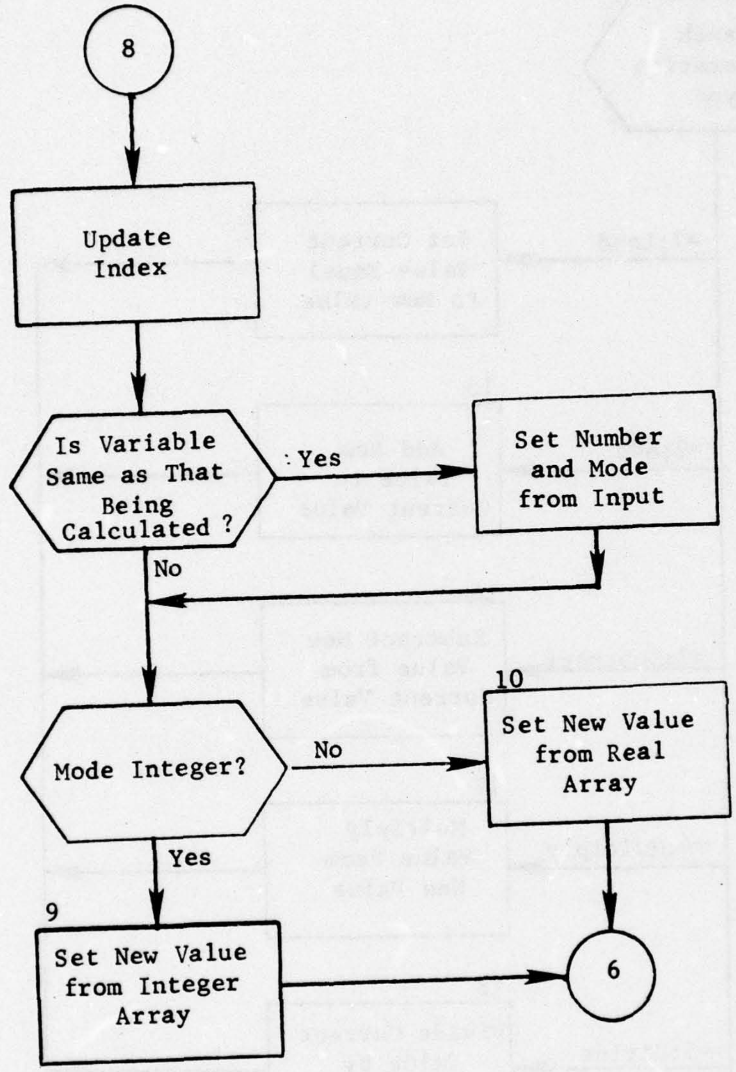


Figure 114. (Part 3 of 4)

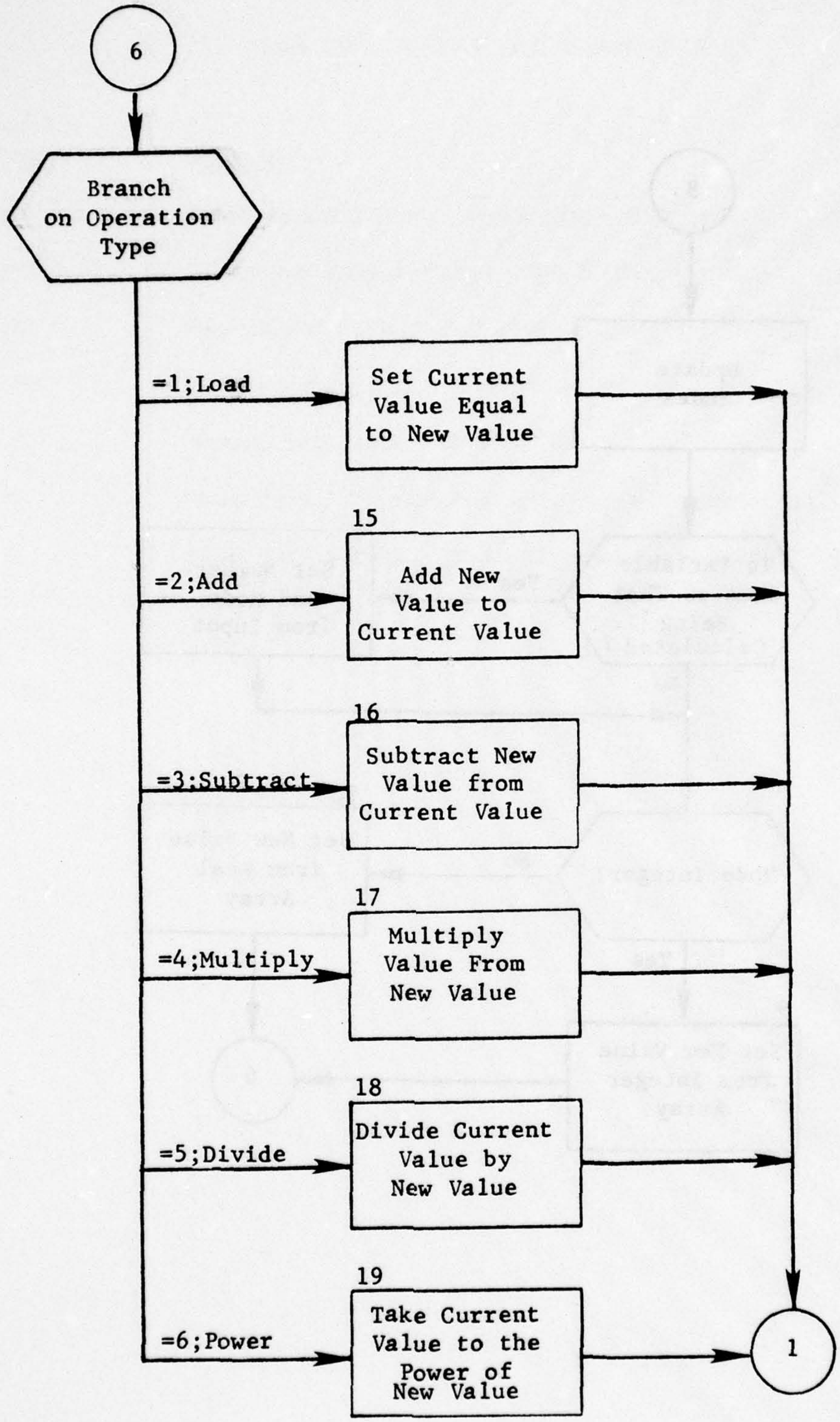


Figure 114. (Part 4 of 4)

SECTION 7. SAVE AND RESTORE MODULE (SRM)

7.1 Purpose

The purpose of the SRM is to give the user the capability to copy the integrated data base onto a magnetic tape and to restore the same data base to a previously stored state by reading in such a tape.

7.2 Input

The input to the SRM is the integrated data base. For a SAVE command, the data base resides on permanent disk files; for a RESTORE command, the data base resides on magnetic tape(s).

7.3 Output

The output of the SRM is the reverse of the input. That is, for a SAVE command, the output is the data base stored onto a magnetic tape; for a RESTORE command the output is the data base stored onto permanent disk files.

7.4 Concept of Operation

SRM uses the SVTP utility subroutine to perform a link-for-link dump or restore of the IDS data file. Since SRM may be called directly by INICOP in the case of a RESTORE verb being the first input, SRM must process any UNIT clause by calling GETSTR directly. This is due to the fact that SRM may be used to restore to an uninitialized file.

7.5 Identification of Subroutine Functions

The entry module of SRM calls no subroutines other than the SVTP utility. This utility is designed to dump or load any random file from tape. The execution of the SVTP utility may result in the special system abort codes listed in table 20.

7.6 Common Blocks

The SRM has no internal common blocks.

Table 20. SVTP System Abort Codes

CODE	MEANING
BC	Block Count Error. Probably a bad tape
IT	Incorrect Device Type. Check JCL
NC	Not Enough Core. Set higher limits
PM	Parameters Mixed. Contact a maintenance programmer
SD	Sequential Disk. Check JCL
ST	Same Type. Sequential unit must be tape
TL	Disk Too Little. Input tape wrong size for disk file

7.7 Subroutine ENTMOD

PURPOSE:	To save and restore the IDS file
ENTRY POINTS:	ENTMOD (first subroutine called when overlay SRM is executed)
FORMAL PARAMETERS:	None
COMMON BLOCKS:	C15, C30, IPQT, OOPS, STRING
SUBROUTINES CALLED:	FILLAD, FILSAV, GETSTR, HDFND, INSGET, RETRV
CALLED BY:	MODGET

Method:

First the ERROR switch is checked. If it is on, the call is from INICOP and special processing must be followed. GETSTR is used to retrieve input strings to look for a possible UNIT clause. If one is found the input unit code is set. Otherwise, POINTR is reset to its original value. ISW is set to 1 since all calls from INICOP are to RESTORE.

If ERROR was not true, INSGET is called for the verb and ISW is set to 1 for RESTORE, 2 for SAVE. Next, the UNIT clause is checked for. If found, the unit is set accordingly.

Then if ISW=1, FILLOD is called, if ISW=2, FILSAV is called. Finally, ERROR is checked again. If it is true it is set to false. If it is false, the index, utility table, and dictionary headers, and the module link tables are retrieved.

Subroutine ENTMOD (SRM) is illustrated in figure 115.

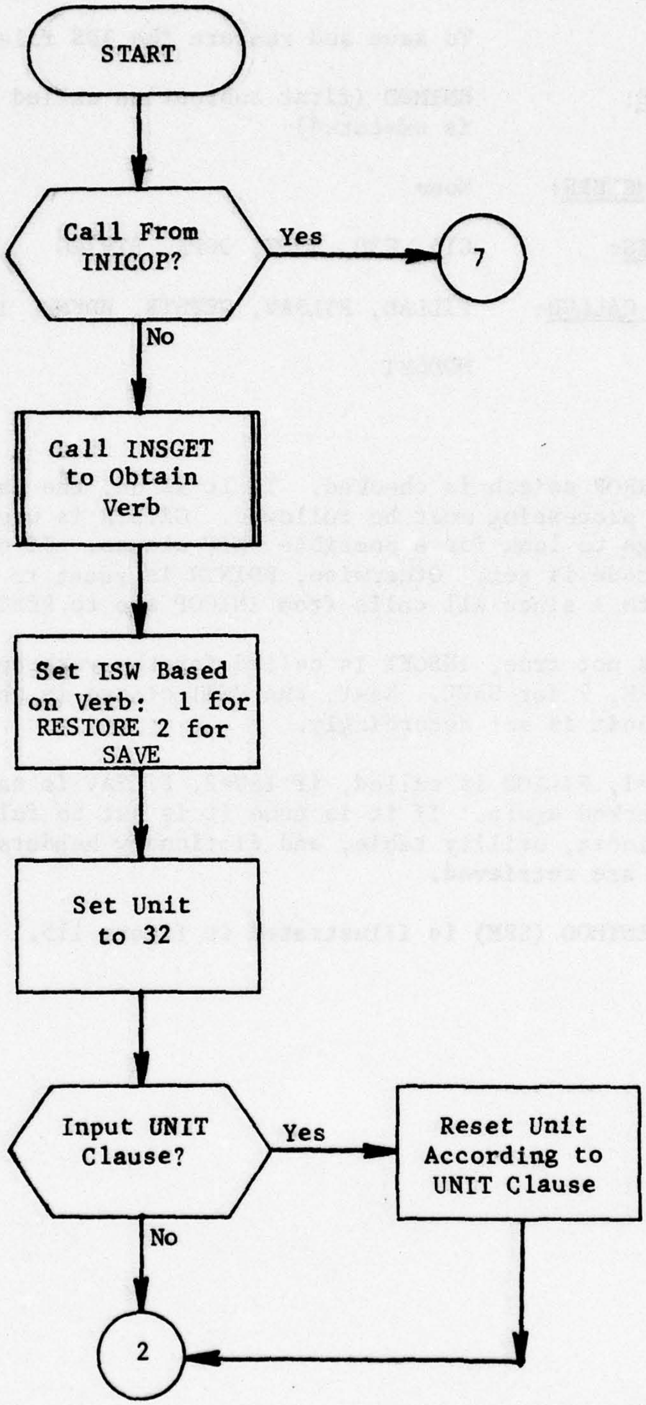


Figure 115. Subroutine ENTMOD (SRM) (Part 1 of 4)

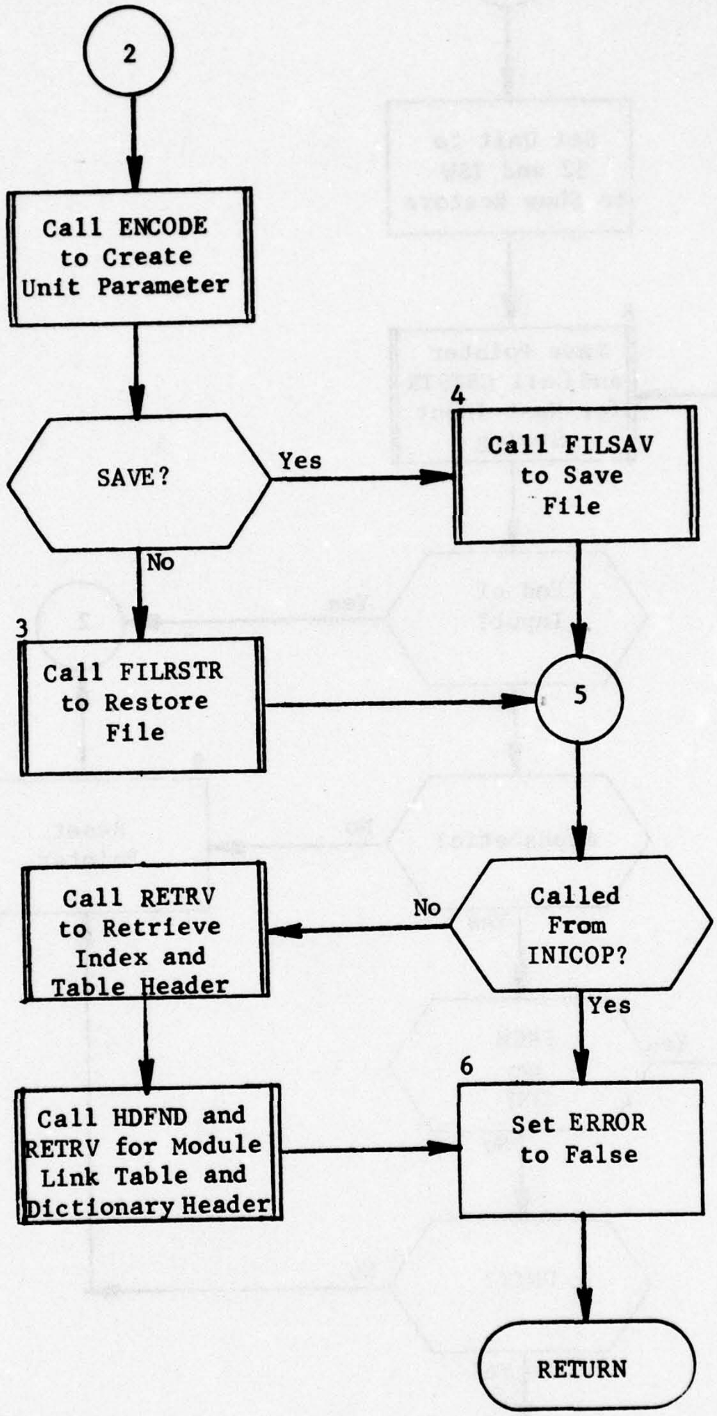


Figure 115. (Part 2 of 4)

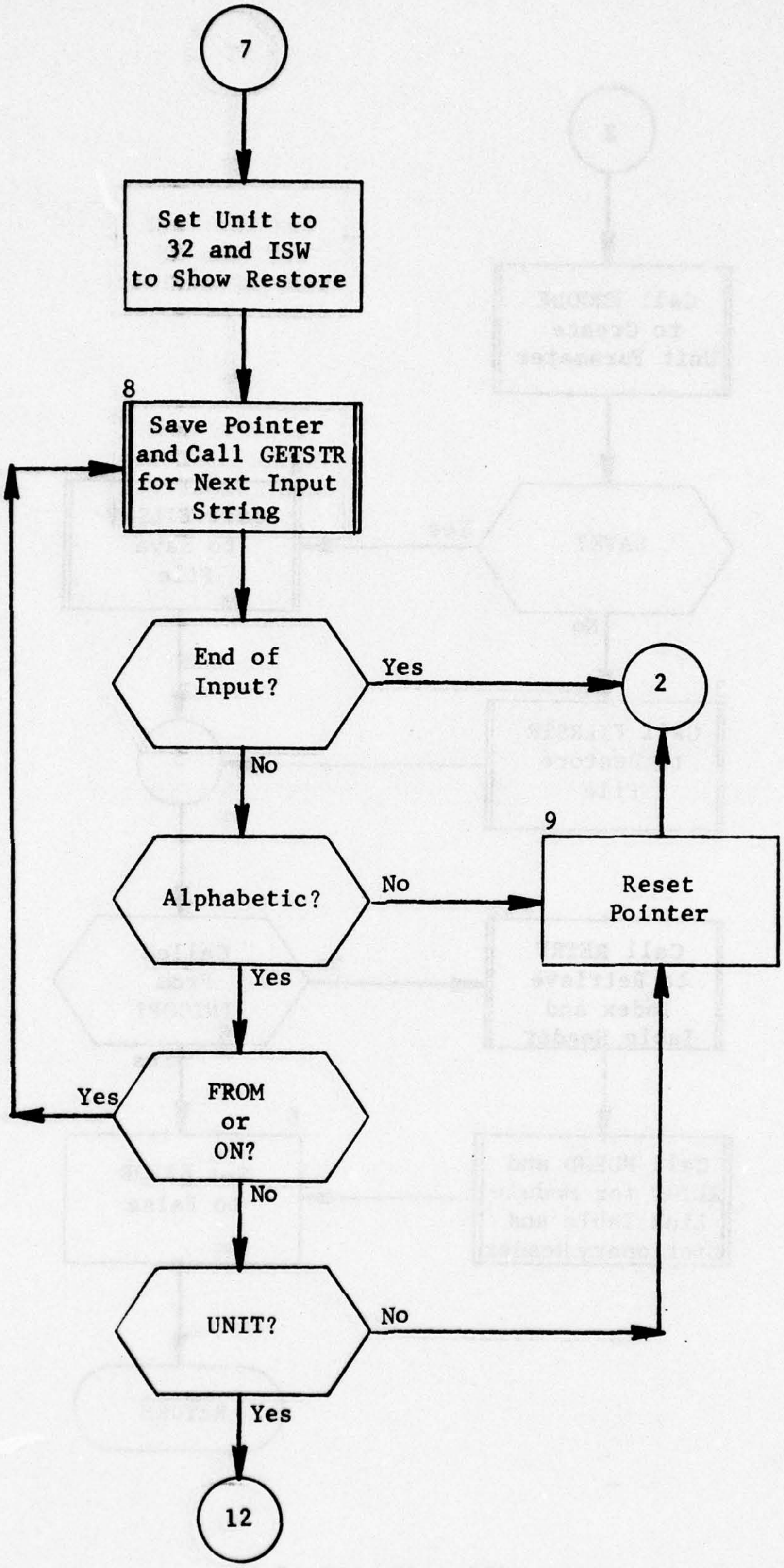


Figure 115. (Part 3 of 4)

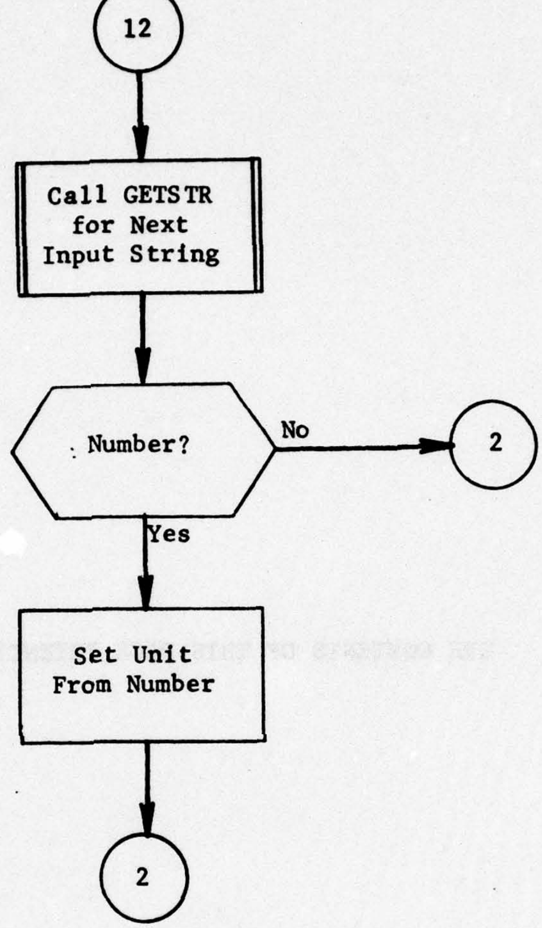


Figure 115. (Part 4 of 4)

Preceding Page Blank - NOT FILMED

SECTION 8. EXTERNAL INTERFACE MODULE (EIM)

8.1 Purpose

The purpose of the EIM is to create output tapes/files which are designed to be input to external processors.

8.2 Input

The output files to be built each have the precondition that all data necessary for the file be present in the data base.

8.3 Output

The output of EIM depends upon the verb and the FILE clause. BUILD FILE TABLE produces a single tape with six subsections. The format of this tape appears in table 21. BUILD FILE SIDAC produces two tapes, one containing BLUE targets, one containing RED targets. The format of the two tapes appears in table 22. BUILD FILE OTHER produces a single tape or file whose format and contents are specified by the user.

The PLOTDATA verb produces two tapes. One is a tape suitable for the CALCOMP plotter. The other contains information concerning all points which were not added to the plot tape owing to their being out of range of the plot size.

8.4 Concept of Operation

The EIM first determines which verb caused the call. If the verb was PLOTDATA, the the PLOTDATA routine is executed. If the verb was BUILD, the FILE clause is found along with the special word within the clause. If the special word is SIDAC or TABLE the appropriate subroutines are called to produce the named files. If the special word is OTHER, the BLDOTH subroutine is called to produce the named files. If the special word is OTHER, the BLDOTH subroutine is called to produce the input defined file.

8.5 Identification of Subroutine Functions

8.5.1 Subroutine SIDAC. This subroutine produces the data base assessment tapes (DBASSESS). A preset retrieval scheme for all BLUE targets is set up and executed. Each target retrieved is written out in a modified JAD format. When all BLUE targets have been output, the scheme is modified for RED targets and the RED targets are written onto a separate tape.

Table 21. BUILD FILE TABLE Output File Formats
(Part 1 of 5)

TARGET LIST

Column	Meaning
1-8	'F1TARGET'
9	Side: 1 for Blue; 2 for Red
10-14	Line Count, numeric
15	Blank
16-20	DESIG, alphabetic
21-24	Blank
25-31	Latitude (LAT), degrees, minutes, seconds
32-39	Longitude (LONG), degrees, minutes, seconds
40	Blank
41-46	NAME, alphabetic
47-50	World Area Code (WACNO), alphabetic
51-55	Bomber Encyclopedia Number (BENO), alphabetic
56	Blank
57-61	Category (CATCODE), numeric
62-63	Country Location (CNTRYL), alphabetic
64-69	Major Complex Number (MAJOR), numeric
70-71	TASK, alphabetic
72-76	Index Number (INDEXNO), numeric
77	Blank
78-80	Complex Number (ICOMPL), numeric

Table 21. (Part 2 of 5)

VEHICLE CHARACTERISTICS LIST

Column	Meaning
1-7	'F1VEHIC'
8	Blank
9	Side: 1 for Blue; 2 for Red
10-14	Line count, numeric
15	'1'
16-20	Missile or bomber line count
21-55	Blank
56-58	CEP in hundreds of feet
59-61	Blank
62-64	CEP in hundreds of feet
65-69	Blank
70-75	TYPE, alphabetic
76-80	Blank

WEAPON CHARACTERISTICS LIST

Column	Meaning
1-8	'F1WEAPON'
9	Side: 1 for Blue; 2 for Red
10-14	Line count
15-17	Blank
18-19	Warhead type (Line Count * 10 plus 1 for ASM 0 for all others)
20	0 = Bomb, 1 = ASM, 2 = DECOY
21-37	Blank
38-43	Warhead yield in kilotons
44-46	Fission to Fusion percentage (FFRAC*100)
47-80	Blank

Table 21. (Part 3 of 5)

MISSILE BASE LIST

Column	Meaning
1-8	'F1MIBASE'
9	Side: 1 for Blue; 2 for Red
10-14	Line Count
15	Blank
16-20	Line Count
21	Blank
22-28	Latitude (LAT) degrees, minutes, seconds
29-36	Longitude (LONG) degrees, minutes, seconds
37	Blank
38-41	Vulnerability Number (VULN1) alphabetic
42-43	Type Count
44-45	Blank
46-47	'1/'
48-49	Number per site (NMPSIT), numeric
50	Blank
51	H if VN greater than or Equal to 20 S otherwise
52	Blank
53	1 if column 51 is H or if 51 is S and NOALER Equal NMPSIT Otherwise = 2
54-59	Blank
60-65	NAME, alphabetic
66-69	Blank
70-71	Country Location (CNTRYL), alphabetic
72-74	Blank
75-80	TYPE, alphabetic

Table 21. (Part 4 of 5)

BOMBER BASE LIST

Column	Meaning
1-6	'FlBASE'
7-8	Blank
9	Side: 1 for Blue; 2 for Red
10	Blank
11-14	Line Count
15	Blank
16-20	Index Number (INDEXNO), numeric
21	Blank
22-28	Latitude (LAT), degrees, minutes, seconds
29-36	Longitude (LONG), degrees, minutes, seconds
37	Blank
38	1 for SLBM, 2 for LRA, 3 for TAC, 7 for all others (from FUNCTI)
39	Blank
40	'X'
41-43	Blank
44	'X' for tanker blank for all others
45-59	Blank
60-65	NAME, alphabetic
66-69	Blank
70-71	Country Location (CNTRYL), alphabetic
72-80	Blank

Table 21. (Part 5 of 5)

OFFENSIVE RECOVERY BASE LIST

Column	Meaning
1-7	'F1RECBS'
8	Blank
9	Side: 1 for Blue; 2 for Red
10-14	Line Count
15	Blank
16-21	DESIG, alphabetic
22-23	Blank
24-30	Latitude (LAT), degrees, minutes, seconds
31-38	Longitude (LONG), degrees, minutes, seconds
39	Blank
40-45	NAME, alphabetic
46-49	World Area Code (WACNO), alphabetic
50-55	Bomber Encyclopedia Number (BENO), alphabetic
56-60	Category Code (CATCODE), numeric
61-62	Country Location (CNTRYL), alphabetic
63-68	Major Complex Number (MAJOR), numeric
69-70	TASK, alphabetic
71-75	Index Number (NDEXNO), numeric
76	Blank
77-80	Capacity (CAPACITY), numeric

Table 22. BUILD FILE SIDAC Output File Format
(Part 1 of 2)

Column	Meaning
1-5	Category code, (CATCODE) numeric
6-9	World Area Code (WACNO) alphabetic
10-15	Bomber Encyclopedia Number (BENO) alphabetic
16-20	Blank
21-26	Name (NAME) alphabetic
27-58	Blank
59-64	Major Complex Number (MAJOR) numeric
65-88	Blank
89-94	Minor Compound Number (MINOR) numeric
95-118	Blank
119-125	Latitude (LAT) degrees, minutes, seconds
126-133	Longitude (LONG) degrees, minutes, seconds
134-137	Blank
138-139	Country Location (CNTRYL) alphabetic
140-147	Blank
148-149	Country Owner (CNTRYO) alphabetic
150-155	Blank
156-159	Severe vulnerability (VULN1) VNTK
160-163	Moderate vulnerability (VULN2) VNTK
164-167	"03PO"
168-190	Blank
191-198	Capacity (POP*10). This quality is zero for all non-U/I targets
199-205	Blank
206-208	Radius (RADIUS*10) numeric - tenth of nautical miles. This quantity is zero for all non-U/I targets
209-288	Blank
289-293	DESIG, alphabetic
294	Blank
295-300	TYPE, alphabetic

Table 22. (Part 2 of 2)

Column	Meaning
301-303	Blank
304-305	ICLASS, numeric
306	1 for Blue targets; 2 for Red targets
307-312	Blank
313-314	TASK, alphabetic
315-318	Blank
319	Region (IREG), numeric
320	SAGA region. This quantity is IREG +1 unless country location is US or AK in which case it is IREG
321-335	Blank
336	Record Mark

8.5.2 Subroutine TABLE. This subroutine produces the table file. The following tables are produced:

- o Target list
- o Vehicle characteristics list
- o Weapon characteristics list
- o Missile base list
- o Bomber base list
- o Offensive recovery base list

For each list a preset retrieval scheme is executed and the records retrieved are written onto the output tape.

8.5.3 Subroutine BLDOTH. This subroutine builds an output file according to user specified formats. This option is very similar in concepts to the REPORT module. First, all input clauses are scanned for attributes and a retrieval scheme is built. Next, all DEFINEs are placed in proper execution order and DEFINE variable execution tables are built. Next, the WHERE clause is scanned for DEFINEs and altered to handle them properly. The sort scheme is now created. Next, the retrieval scheme is executed and a file built of the resultant records and this file is sorted. Finally, the FORMAT clause is executed to produce the desired file (see figure 116).

8.5.4 Subroutine PLOTDATA. This subroutine builds an output plot tape. Four types of geographic data may be plotted:

- o Penetration corridors
- o Depenetration corridors
- o Refuel points
- o Recovery bases

For each desired set of data a preset retrieval scheme is used to retrieve the appropriate data. For each data record retrieved, the coordinates (LAT, LONG) are converted and/or scaled to the desired map characteristics and are processed for the plotter.

8.6 Common Blocks

The common blocks internal to EIM are listed in table 23.

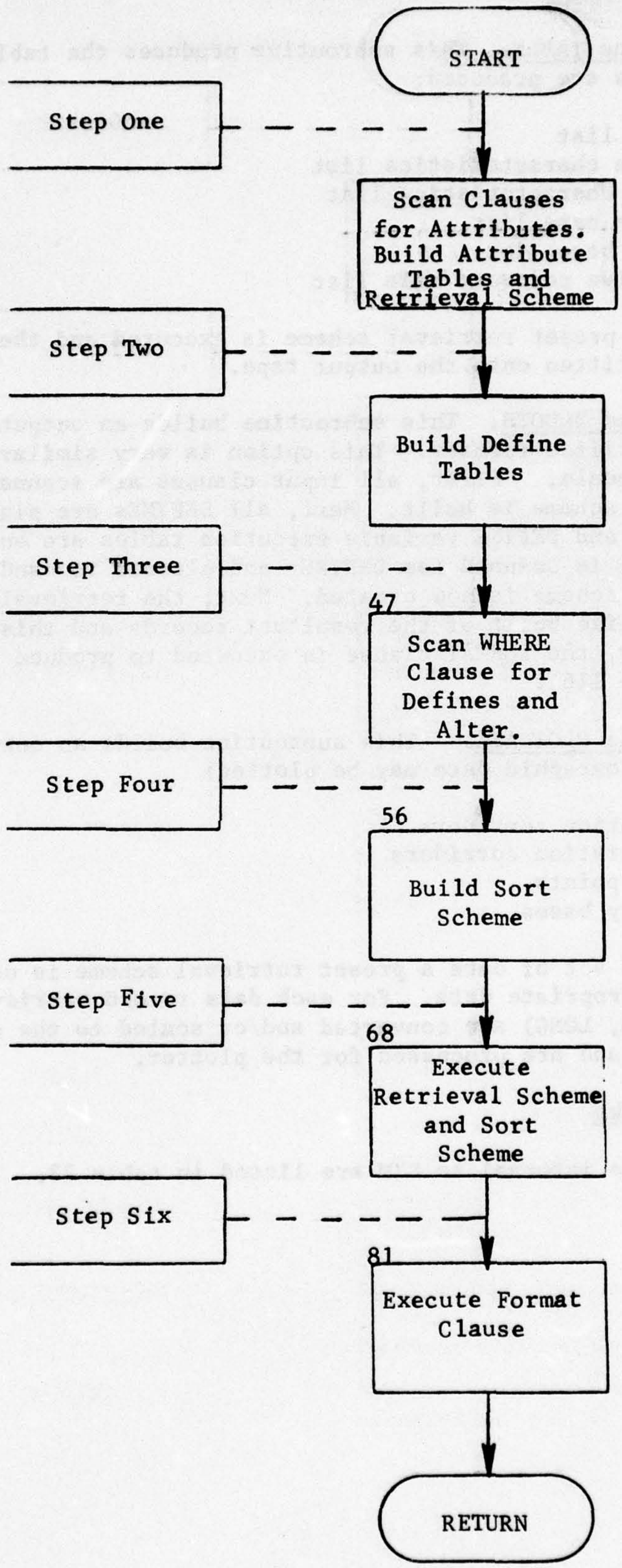


Figure 116. Subroutine BLDOTH: Macro Flow

Table 23. EIM Internal Common Blocks (Part 1 of 3)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
ATLST		Provides communication with the ATFNDR utility subroutine
	ATNUMB(100)	Attribute's identifying number
	ATADD(100)	Attribute's address
	ATTYP(100)	Attribute's mode (=1, integer, =2, alphabetic, =3, floating point)
	ATRA(100)	Attribute's lower limit
	ATRB(100)	Attribute's upper limit
	NUMAT	Number of attributes
DEFNMZ	DFNAME(100)	DEFINE variable name
	DFPNT(100)	Points to DEFINE clause in the Instruction code
DEFVAR	VARXX(100)	Value of DEFINE variable
DSPFRM		Provides communication with FORMAK utility subroutine
	FORMAR(50)	Format being constructed
	IFPNT	Pointer to next character
	IFLNG	Number of words used
PLTPRO	MERCAT	Project type (set to 0)
	IDIREC	Indicator of plot direction (+1 for counter clockwise, -1 for clockwise)
	FLMDAO	Longitude of origin
	PHIO	Latitude of origin
	THETAO	Angle between meridian and X-axis
	PHI1	Standard parallel closest to equator
	PHI2	Standard parallel closest to pole
	A1L10	Fractional part of log for PHI1
	A2L10	Fractional part of log for PHI2

Table 23. (Part 2 of 3)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
PLTSPE	ISZE	Plot size indicator =0 for 50 x 40 =1 for 20 x 20 =2 for 10 x 10
PLTSPE	XAXLEN	Length of X-axis in inches
	YAXLEN	Length of Y-axis in inches
	FACTOR	Number of plots per page
	SCALE	Ratio of world units to plot units
PRINSP	PRINON	Switch to control optional print =True, produce print =False, do not produce print
PSCOM		Used to communicate with PSREC utility
	RECORD(100)	Body of print/sort record
	RECLEN	Number of words in record
RTLST		Used to communicate with ATFNDR utility
	RTLIST(100)	List of record type numbers for retrieval
	NUMREC	Number of record types in list
	HDREC	Record type name of primary header
	HCLASS	CLASS value of primary header
	HSIDE	SIDE value of primary header
	HOPT	CLASS/SIDE option for scheme
	JHDR	Record type number of primary header
SCHEME	POINT	Pointer to current instruction of retrieval scheme
	SCHEME(200)	Retrieval scheme (see UM I, section 5.3.2.4)
SORSCH	SRTSCH(100)	Sort scheme (see section 6.10)
TAPES	PLOTTA	Logical unit number for plot tape
	PIECTA	Logical unit number for tape for nonplotted points

Table 23. (Part 3 of 3)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
XMEDGE	XMEDGE	Map edge
	XLL	X-coordinate of last point
	YLL	Y-coordinate of last point
	XL	X-coordinate of point to be plotted
	YL	Y-coordinate of point to be plotted
	XWEDGE	Converted value for latitude of origin
	BANGL	Converted value for longitude of origin
ZEES		Used internally for core position only
	ZA, ZB, ZC, ZD, ZE	Equivalent to output array from INSGET

8.7 Subroutine ENTMOD

PURPOSE: Entry subroutine for EIM

ENTRY POINTS: ENTMOD (first subroutine called when overlay EIM is executed)

FORMAL PARAMETERS: None

COMMON BLOCKS: OOPS, PRINSP

SUBROUTINES CALLED: BLDOTH, INSGET, PLOTDA, SIDAC, TABBLE

CALLED BY: MODGET

Method:

First the input is scanned for the ONPRINTS adverb and if found the PRINON switch is set to true. Next, the verb is retrieved, if it is PLOTDATA the overlay link and PLOTDA are executed. If the verb is BUILD, the FILE clause is found and the special word checked. If it is TABLE, the overlay for TABBLE is called. If it is SIDAC, the overlay link for SIDAC is called. If it is OTHER the overlay link for BLDOTH is called.

Subroutine ENTMOD (EIM) is illustrated in figure 117.

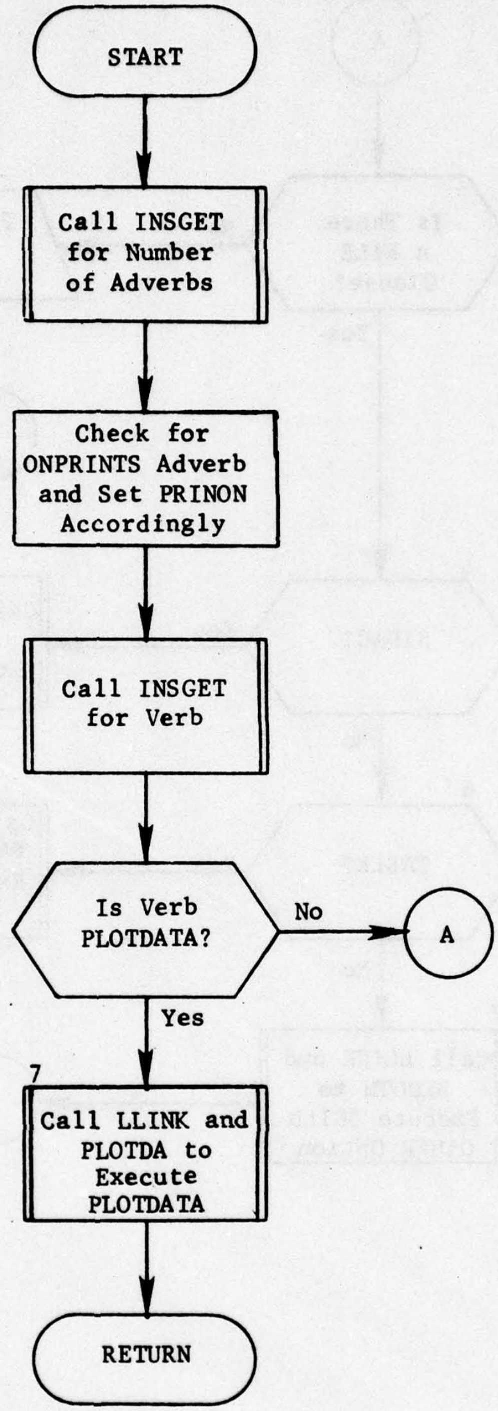


Figure 117. Subroutine ENTMOD (EIM) (Part 1 of 2)

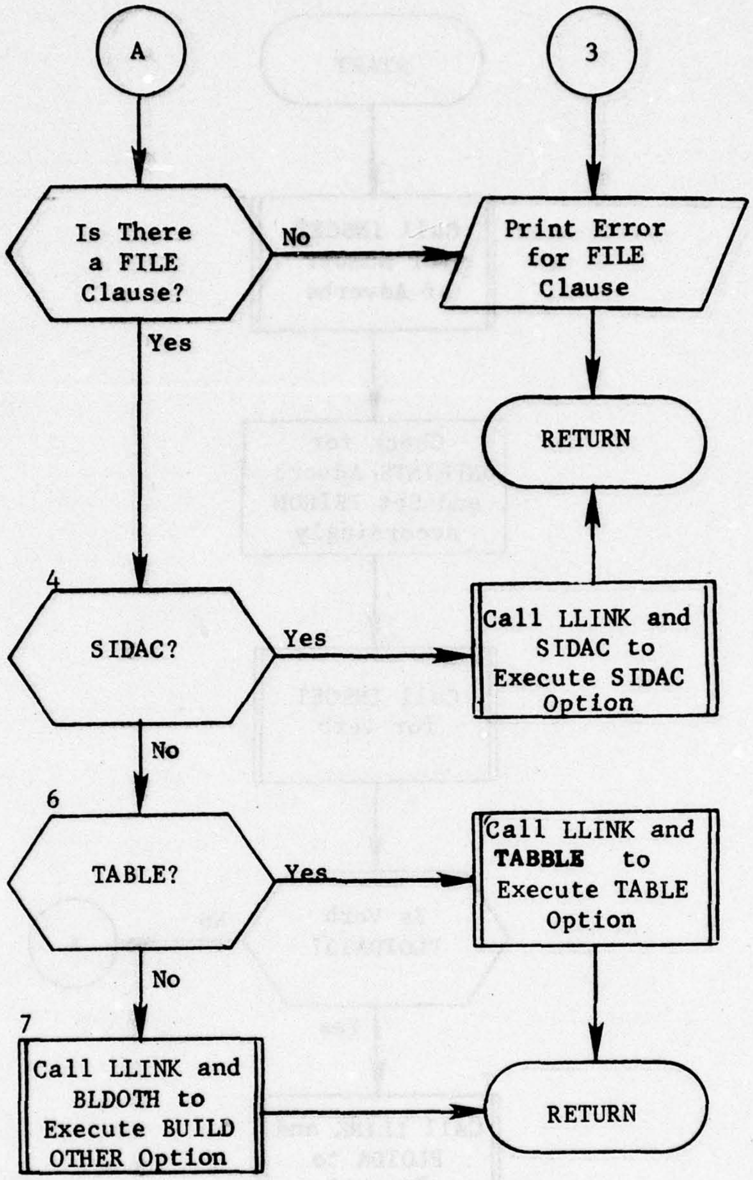


Figure 117. (Part 2 of 2)

8.7.1 Subroutine CONVLL

PURPOSE: Convert latitude and longitude to degrees, minutes, seconds (DMS) format

ENTRY POINTS: CONVLL

FORMAL PARAMETERS:

	XLAT:	Input latitude
	XLONG:	Input longitude
	CHLAT:	Output latitude (character *7)
	CHLONG:	Output longitude (character *8)

COMMON BLOCKS: None

SUBROUTINES CALLED: None

CALLED BY: SIDAC, TABBLE

Method:

The process is similar for both latitude and longitude. The latitude is converted first. The letter (CHM) is set to N and if the latitude is negative it is set to positive and CHM is set to S. The degrees, minutes and seconds are then broken out and ENCODEd into CHLAT. Longitude is now processed, CHM is set to W. If longitude is greater than 180 it is subtracted from 360 and CHM is set to E. Longitude is then broken down and ENCODED into CHLONG.

Subroutine CONVLL is illustrated in figure 118.

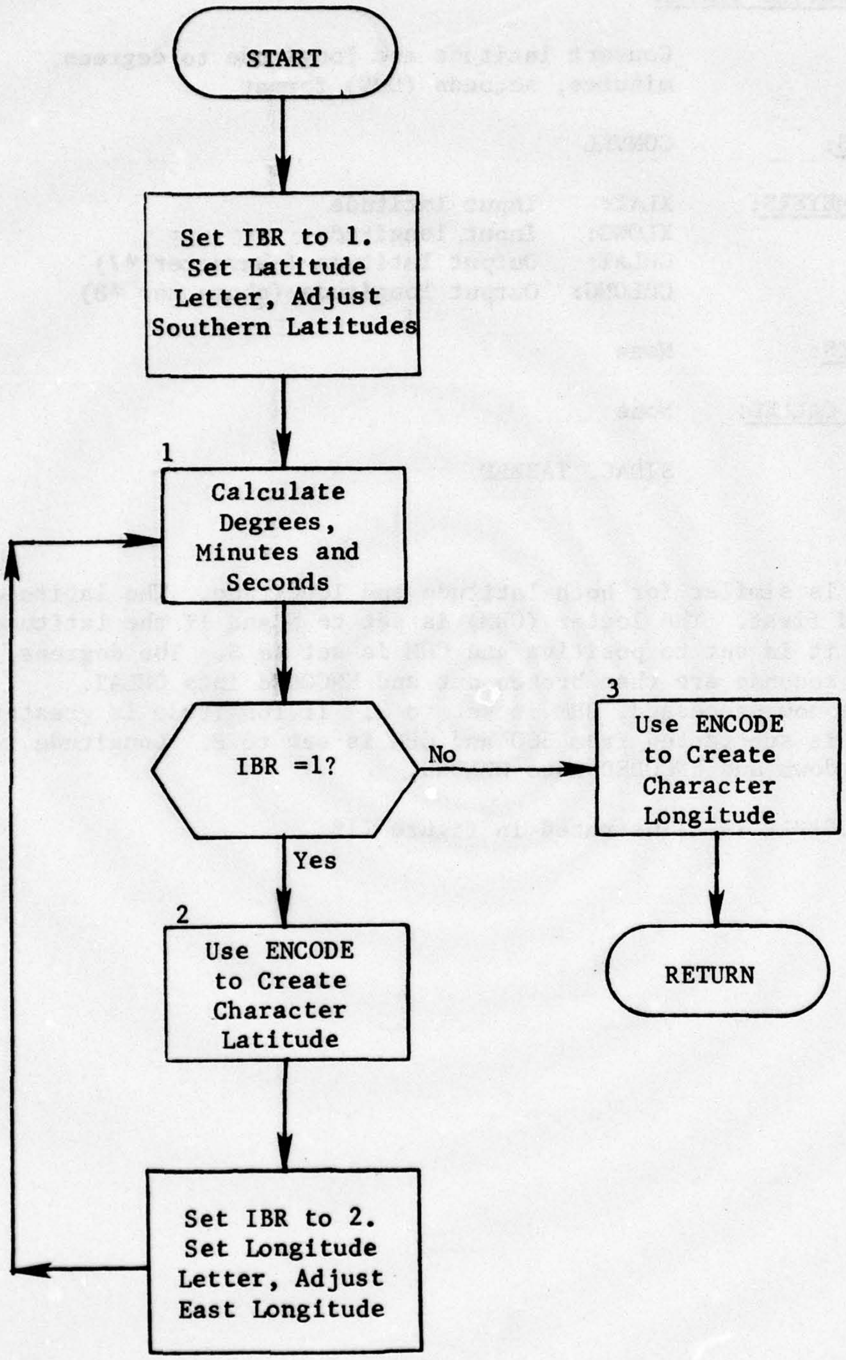


Figure 118. Subroutine CONVLL

8.8 Subroutine BLDOTH*

PURPOSE: To construct an input defined file

ENTRY POINTS: BLDOTH

FORMAL PARAMETERS: None

COMMON BLOCKS: ATLST, C30, DEFNMZ, DEFVAR, DSPFRM, OOPS, PSCOM, RTLST, SORSCH, ZEES

SUBROUTINES CALLED: ATFNDR, FORMAK, GETNXT, INSGET, INSPUT, LINKUP, PSNXT, PSPUT, PSREC, PSRWD, SETSCH, UNCODE, XDEFN, XSORT, XWHERE

CALLED BY: ENTMOD (EIM)

Method:

Step One

If there is a UNIT clause it is used to reset the output unit (default =35). Next the DEFINE, WHERE, SORT, and FORMAT clauses are all scanned for attributes. The attributes in the SORT and FORMAT clauses are flagged as being part of the print (sort record (ATCLZ=4 or 5). Also, the WHERE clause is scanned for values for the SIDE or CLASS attributes and the names of the DEFINEs are saved (see figure 119).

Step Two

The set of DEFINE clauses is processed to build the DEFINE tables (similar to element 2-7 in table 18). The DEFINEs are placed in the table in the order in which they should be executed. This order is dependent upon the presence with the execution instructions of other DEFINEs. These other DEFINEs have to be executed prior to the DEFINE which contains them. The process determines the mode (DFNMOD) and type (DFNTYP) of the DEFINE. Normal DEFINEs are also assigned positions in the print/sort record (DFNPOS). The alphabetic instructions which represent a DEFINE are altered in the following way:

- o The second word is changed from 9 to 8
- o The third word is set to the DEFINEs ordered number
- o The fourth word is set to the DEFINEs mode

(If the DEFINE instruction being altered is one whose name is the same as the DEFINE clause which contains it, the third and fourth words are set to zero) (see figure 120).

*Main routine of overlay BOTHER

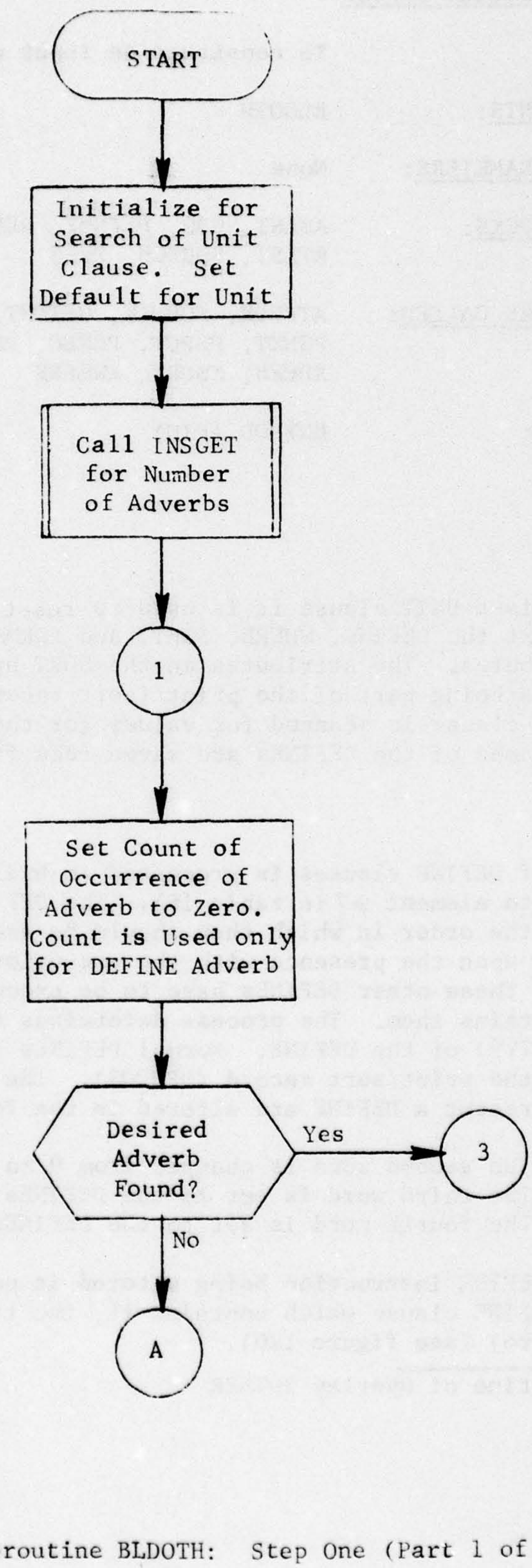
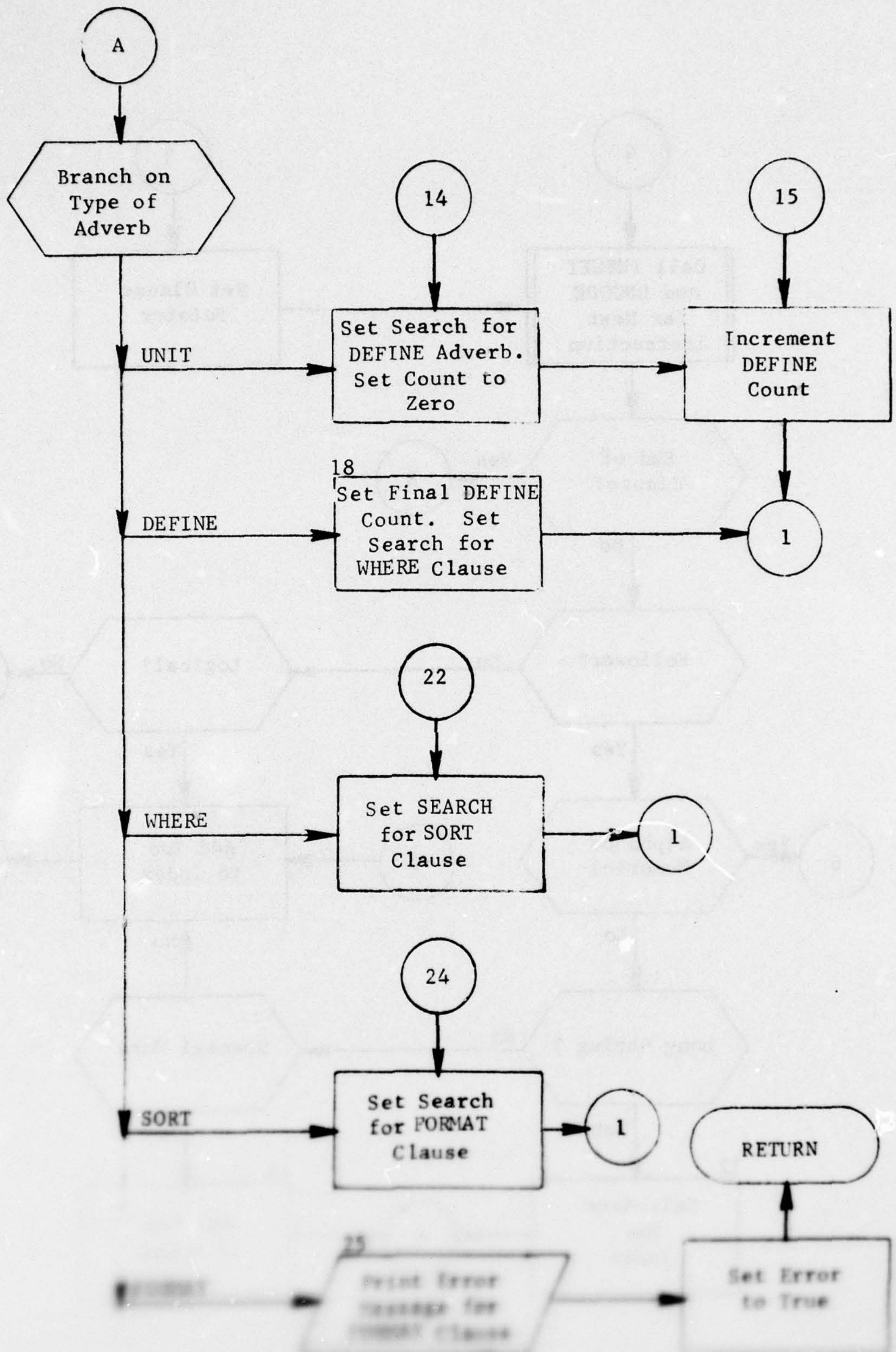


Figure 119. Subroutine BLDOTH: Step One (Part 1 of 7)



A
Branch on Type of Adverb
UNIT
DEFINE
WHERE
SORT
14
Set Search for DEFINE Adverb. Set Count to Zero
15
Increment DEFINE Count
18
Set Final DEFINE Count. Set Search for WHERE Clause
1
22
Set SEARCH for SORT Clause
1
24
Set Search for FORMAT Clause
1
RETURN
25
Set Error to True

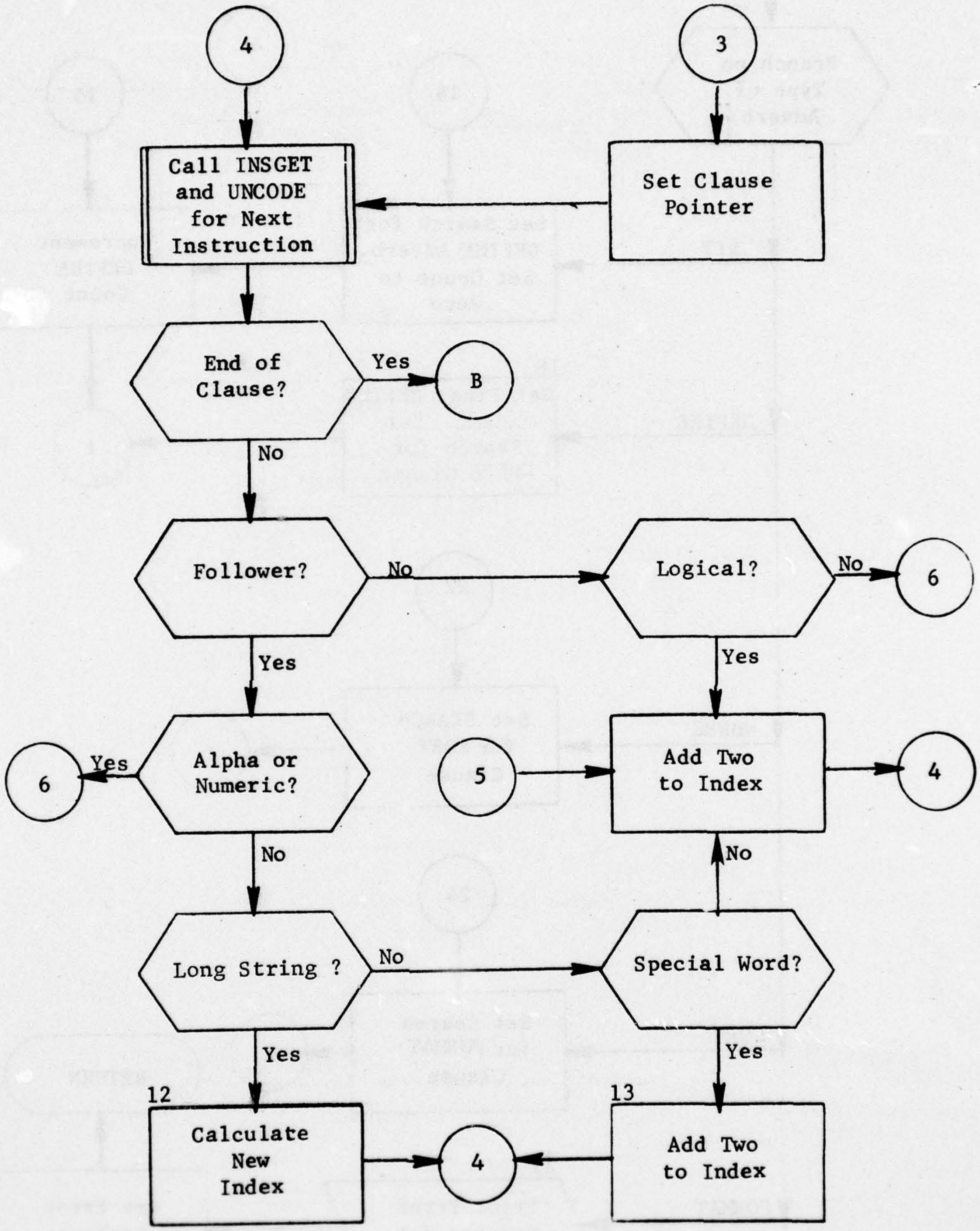


Figure 119. (Part 3 of 7)

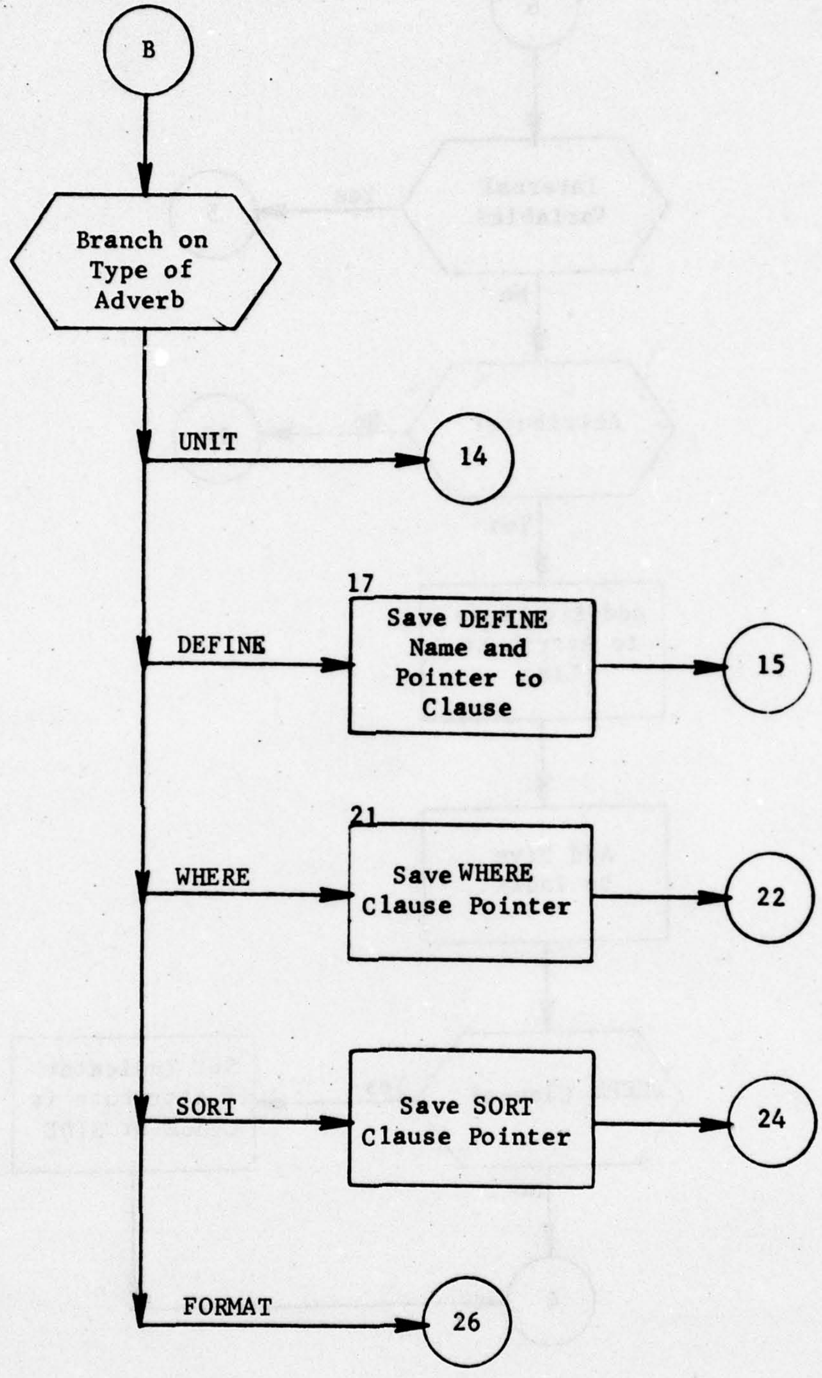


Figure 119. (Part 4 of 7)

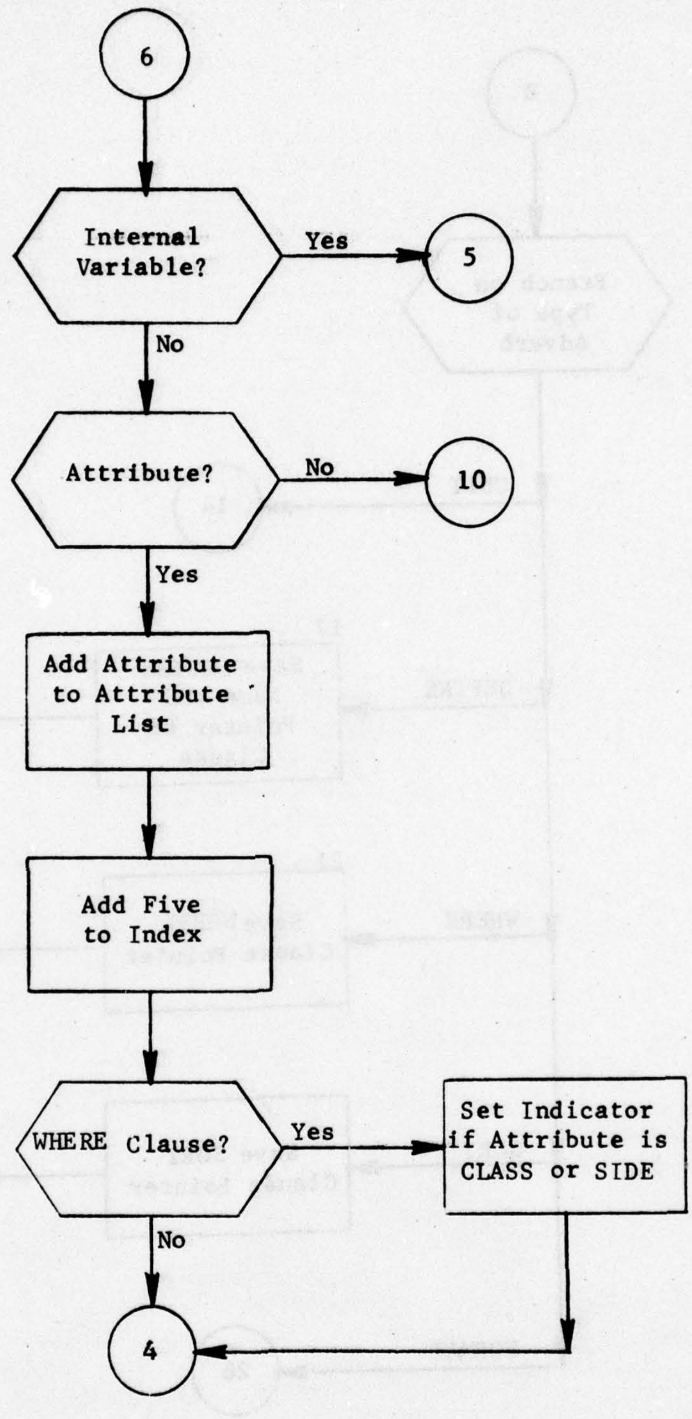


Figure 119. (Part 5 of 7)

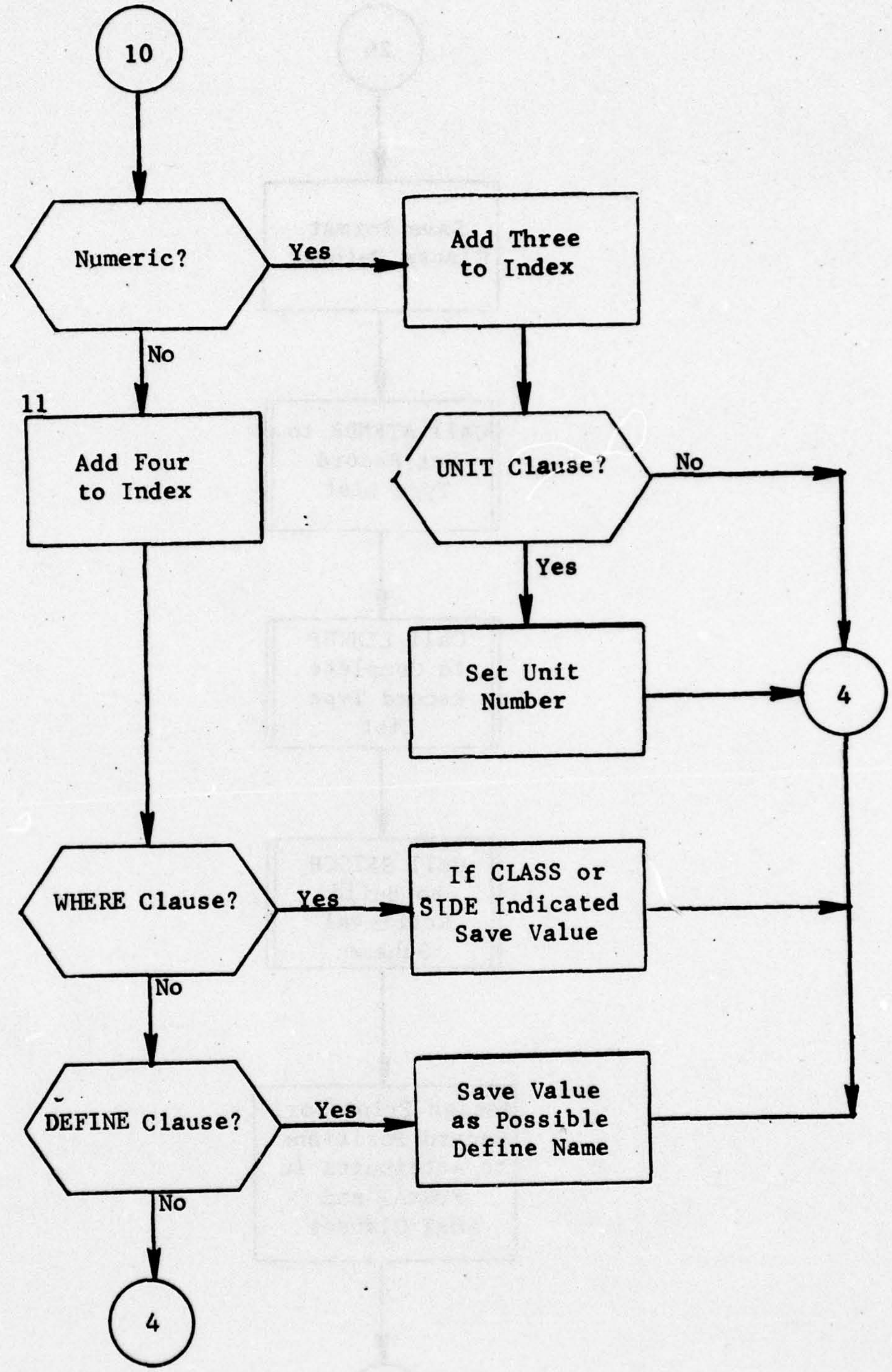


Figure 119. (Part 6 of 7)

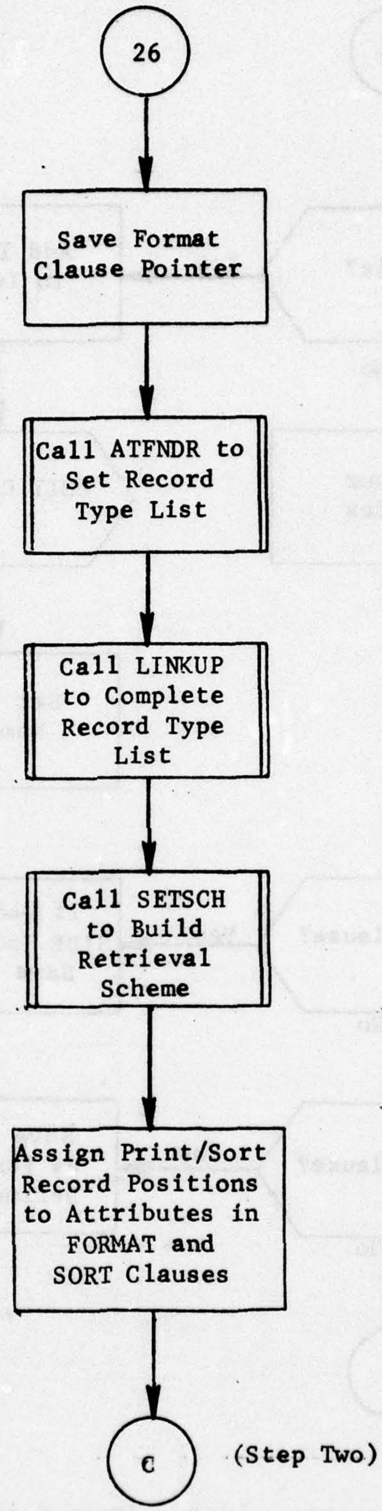


Figure 119. (Part 7 of 7)

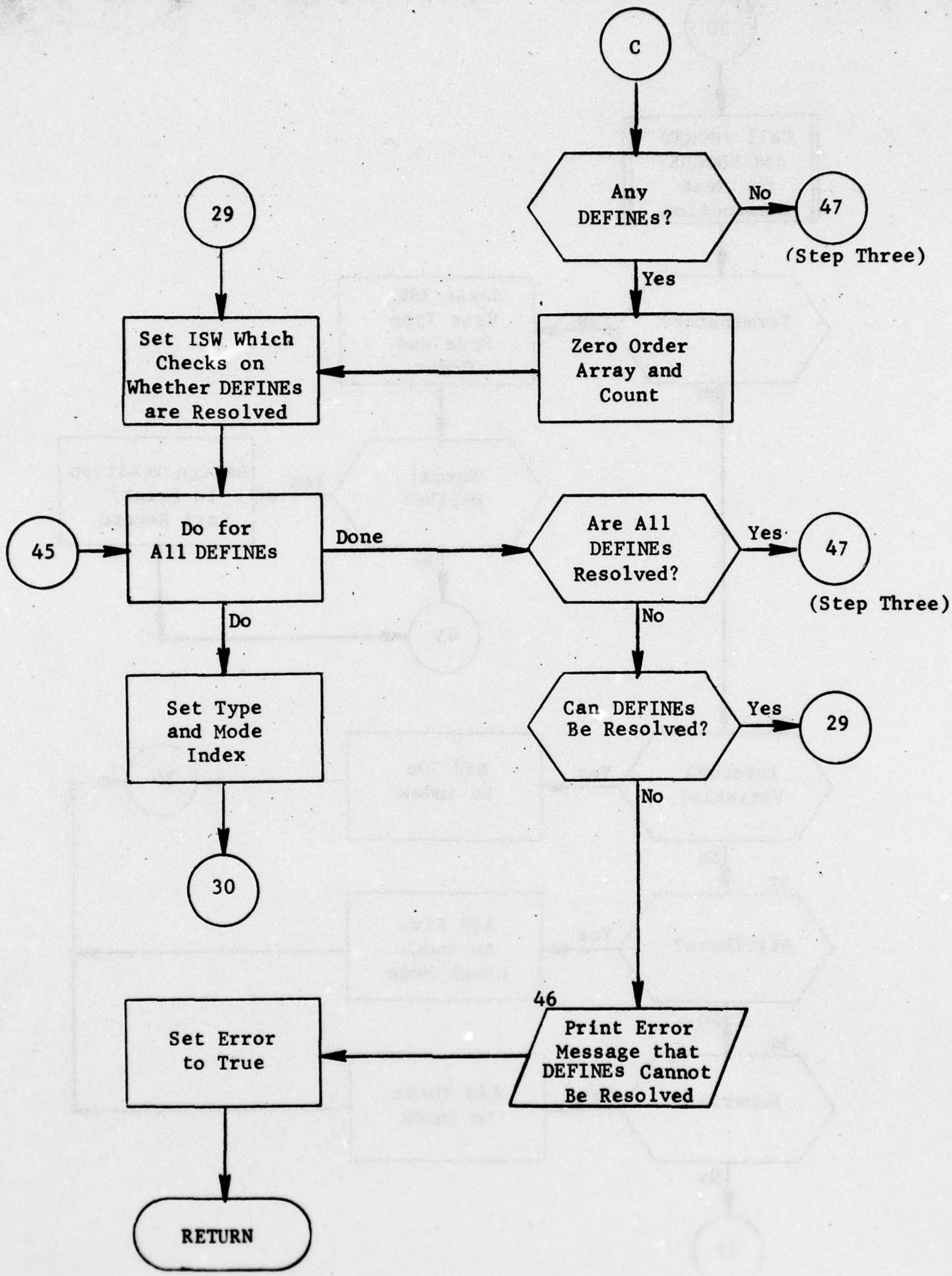


Figure 120. Subroutine BLDOTH: Step Two (Part 1 of 4)

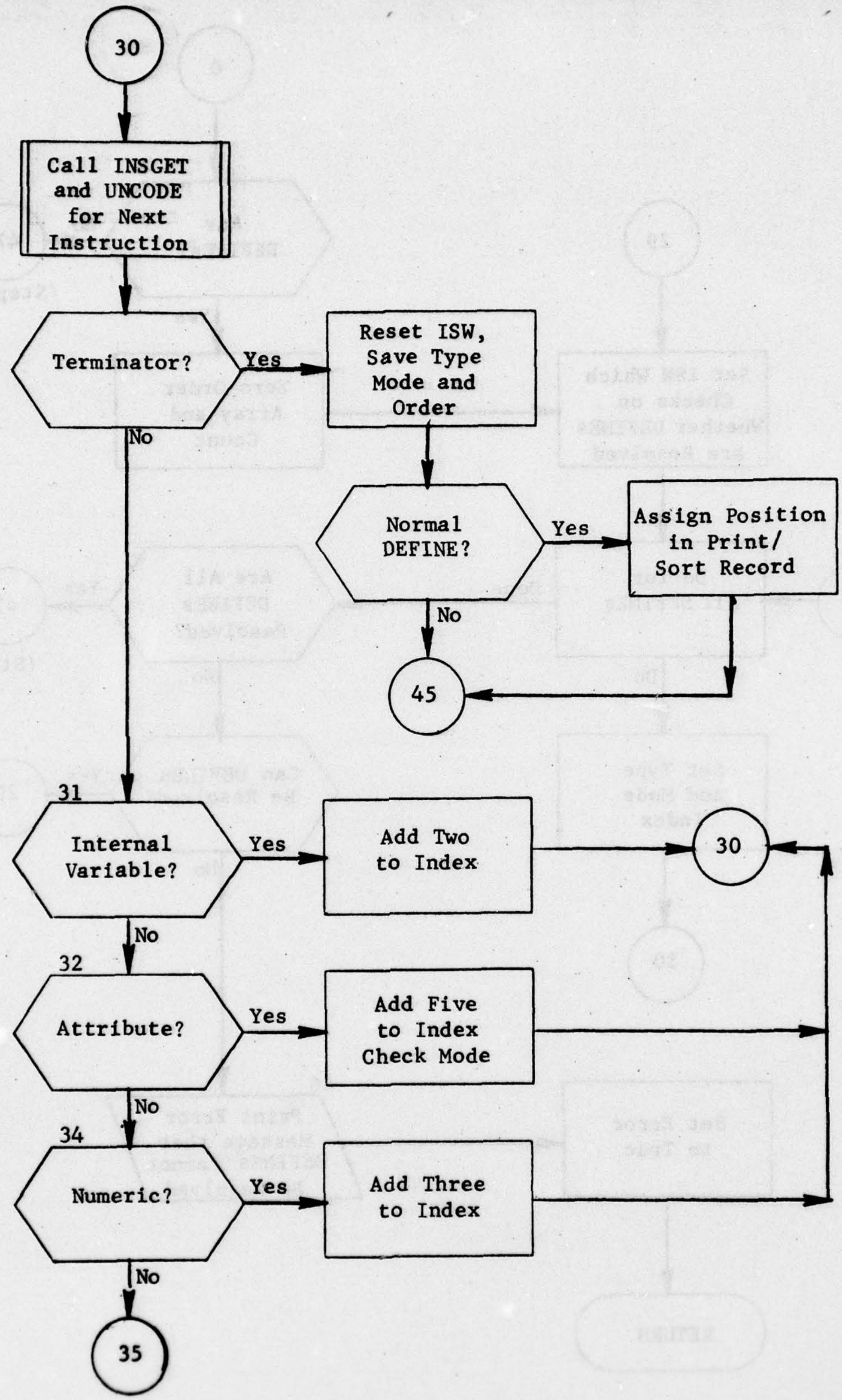


Figure 120. (Part 2 of 4)

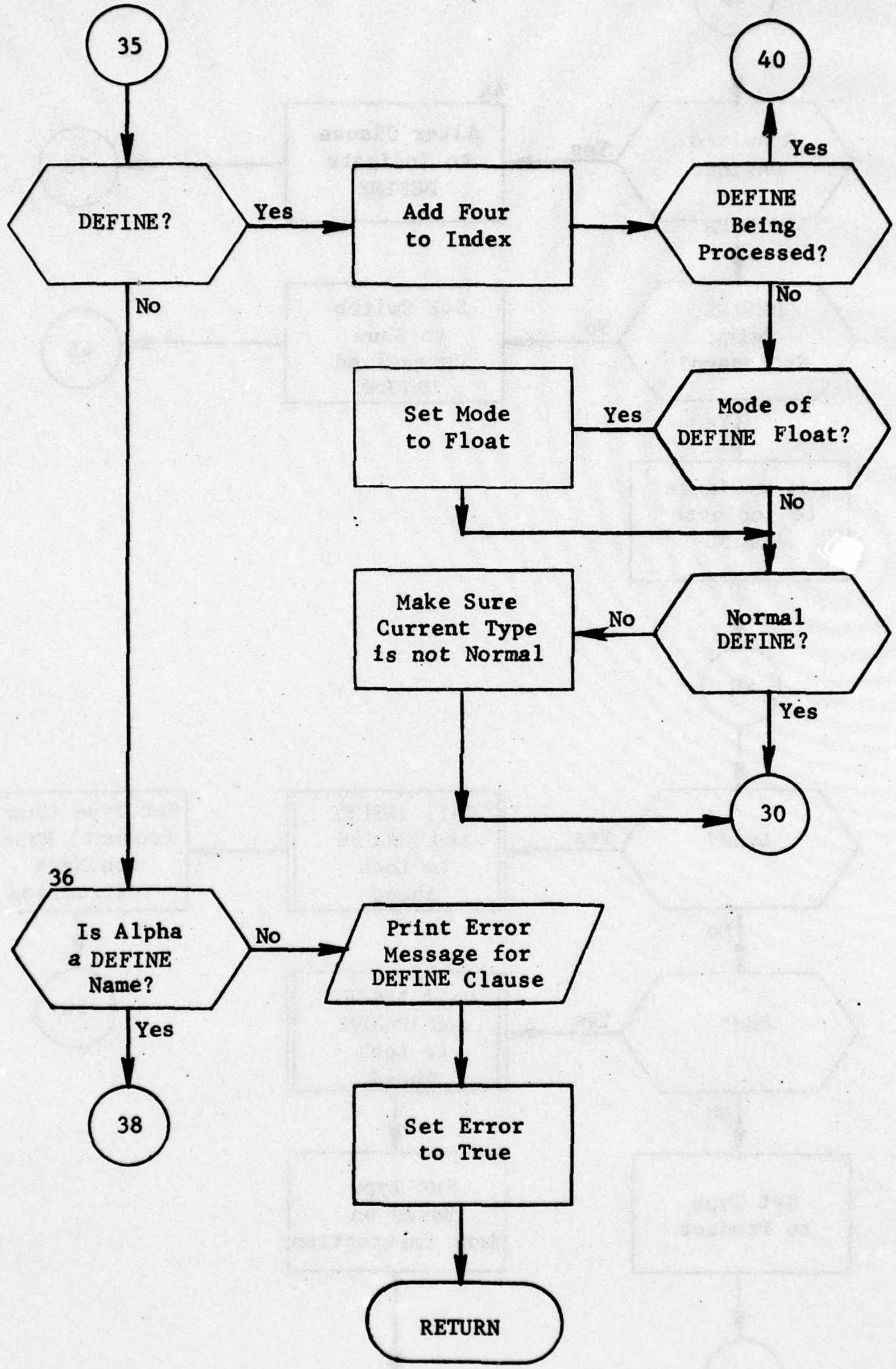


Figure 120. (Part 3 of 4)

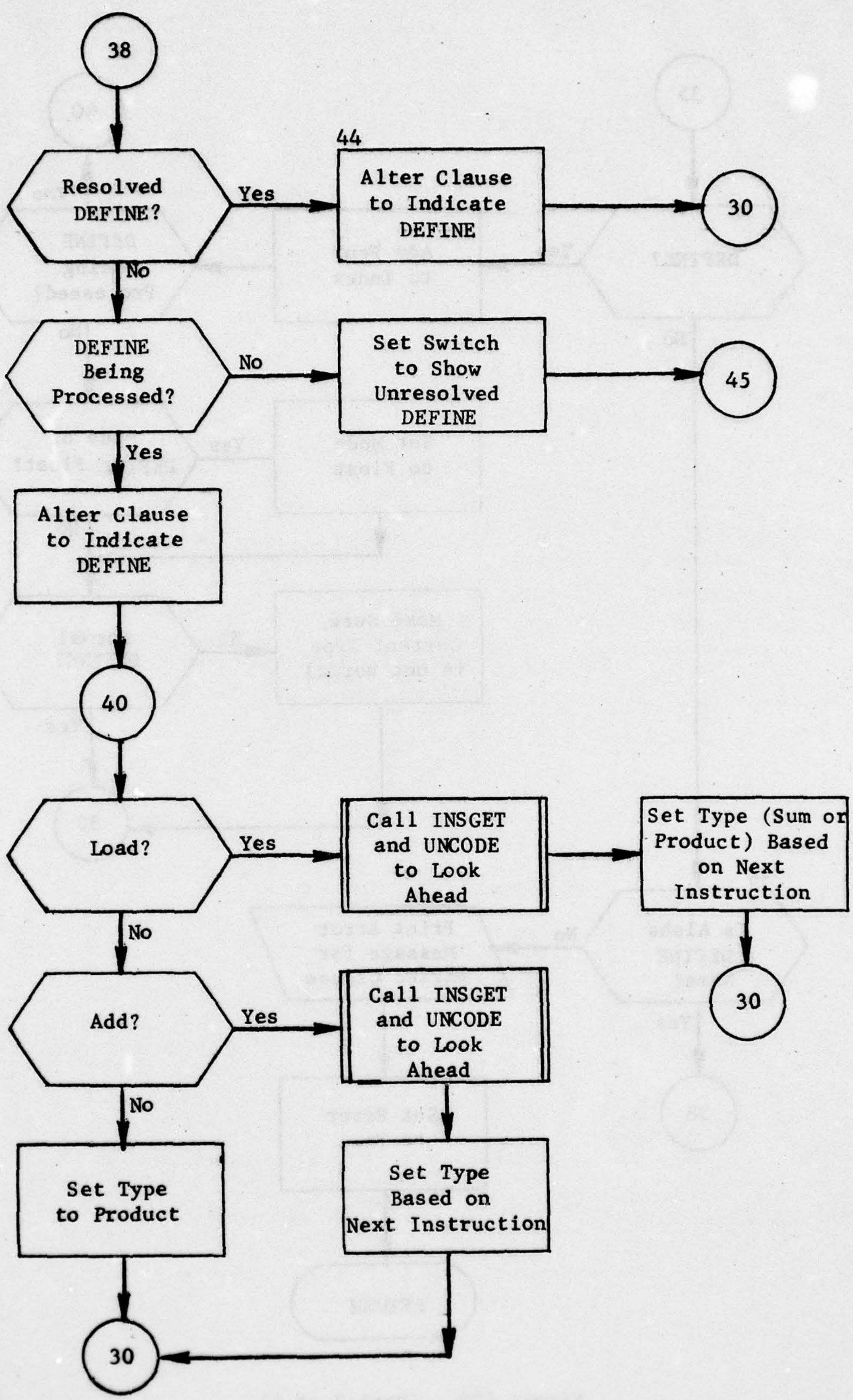


Figure 120. (Part 4 of 4)

Step Three

The WHERE clause is now scanned for alphabetic instructions that represent DEFINEs and these instructions are altered as per step two (see figure 121).

Step Four

The SORT clause is now retrieved in pairs of instructions. Each pair should contain an attribute or DEFINE name plus one of the special words: ASCENDING(A) or DESCENDING(D). These pairs are used to create the sort scheme in common block SORSCH similar to element 10 of table 18 (see figure 122).

Step Five

First PSREC is called and all sum and product DEFINEs are initialized. Next GETNXT is called to execute the retrieval scheme. For each return from GETNXT, XDEFN is called to execute all DEFINEs and XWHERE is called to perform selection. If a record is selected, the appropriate attributes and DEFINEs are stored in the RECORD array and PSPUT is called to add a new print/sort record. Finally, when the retrieval scheme is complete, XSORT is called to sort the print/sort file (see figure 123).

Step Six

In this step, the FORMAT clause is processed and the output file produced. The FORMAT clause must be made up of PAGE and/or LINE items. When a PAGE item is encountered a file mark is written on the tape. When a LINE item is encountered every element in it is scanned. Attributes and DEFINE names are added to print element tables (see discussions on print schemes 6.4) and each element causes additions to the format being built by FORMAK. When the entire LINE item has been scanned, the process depends on whether the LINE contained as elements any attributes or normal DEFINEs. If not, a single record is produced with the created format. If so, the process begins with a call of PSRWD and the initialization of sums and products. Then PSNXT is called for each print/sort record. When it is retrieved, XDEFN is called for all DEFINEs that are not normal. Finally a record is produced using the created format for each print/sort record. The above process continues until all items in the FORMAT clause have been processed. An end of file is then written on the output unit and it is rewound (see figure 124).

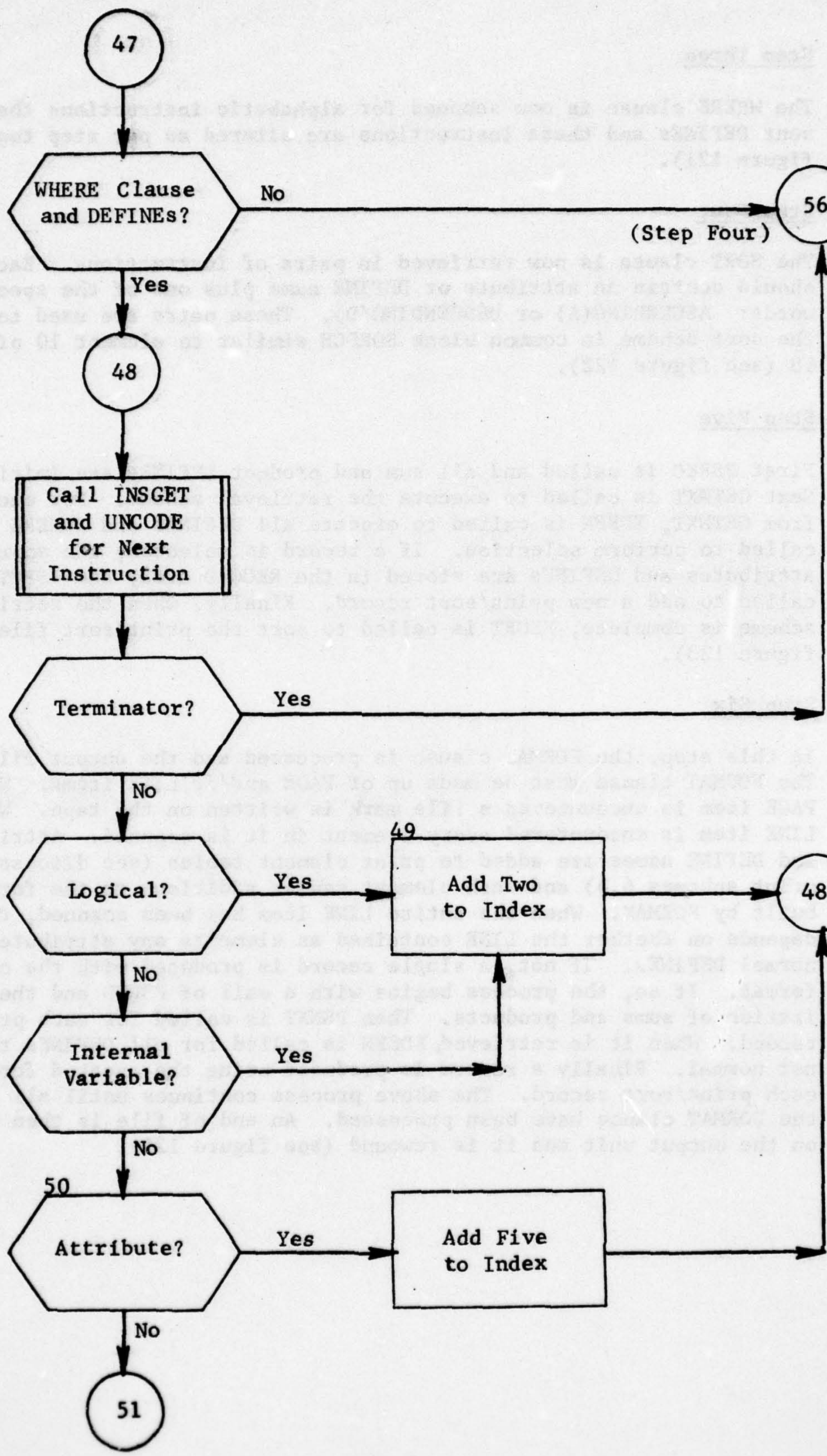


Figure 121. Subroutine BLDOTH: Step Three (Part 1 of 2)

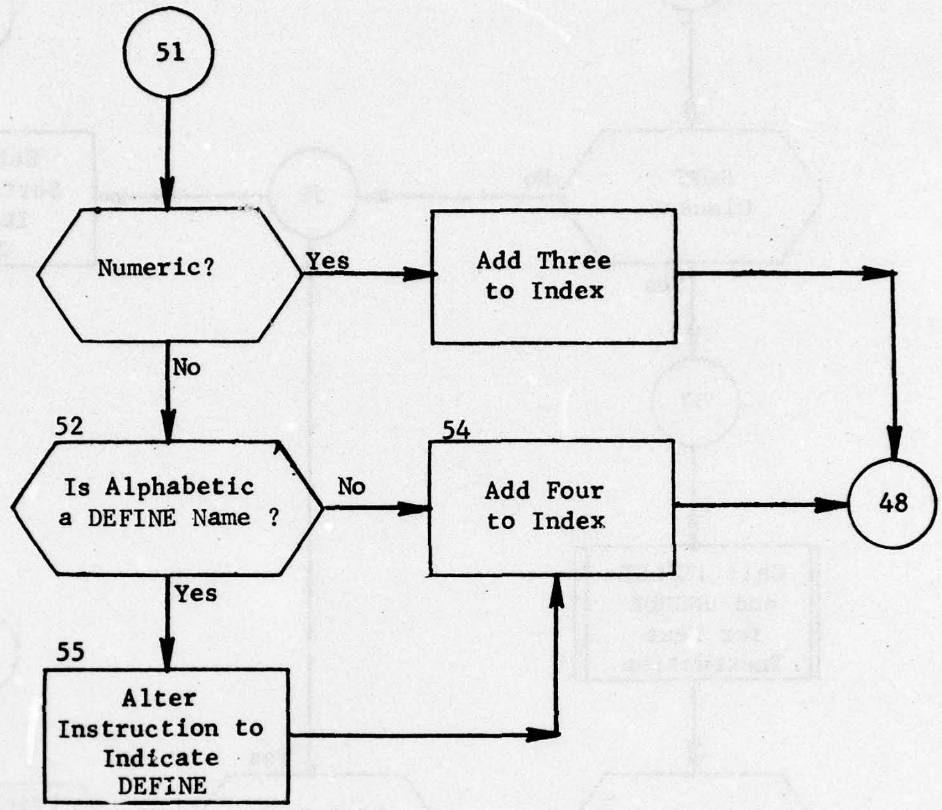


Figure 121. (Part 2 of 2)

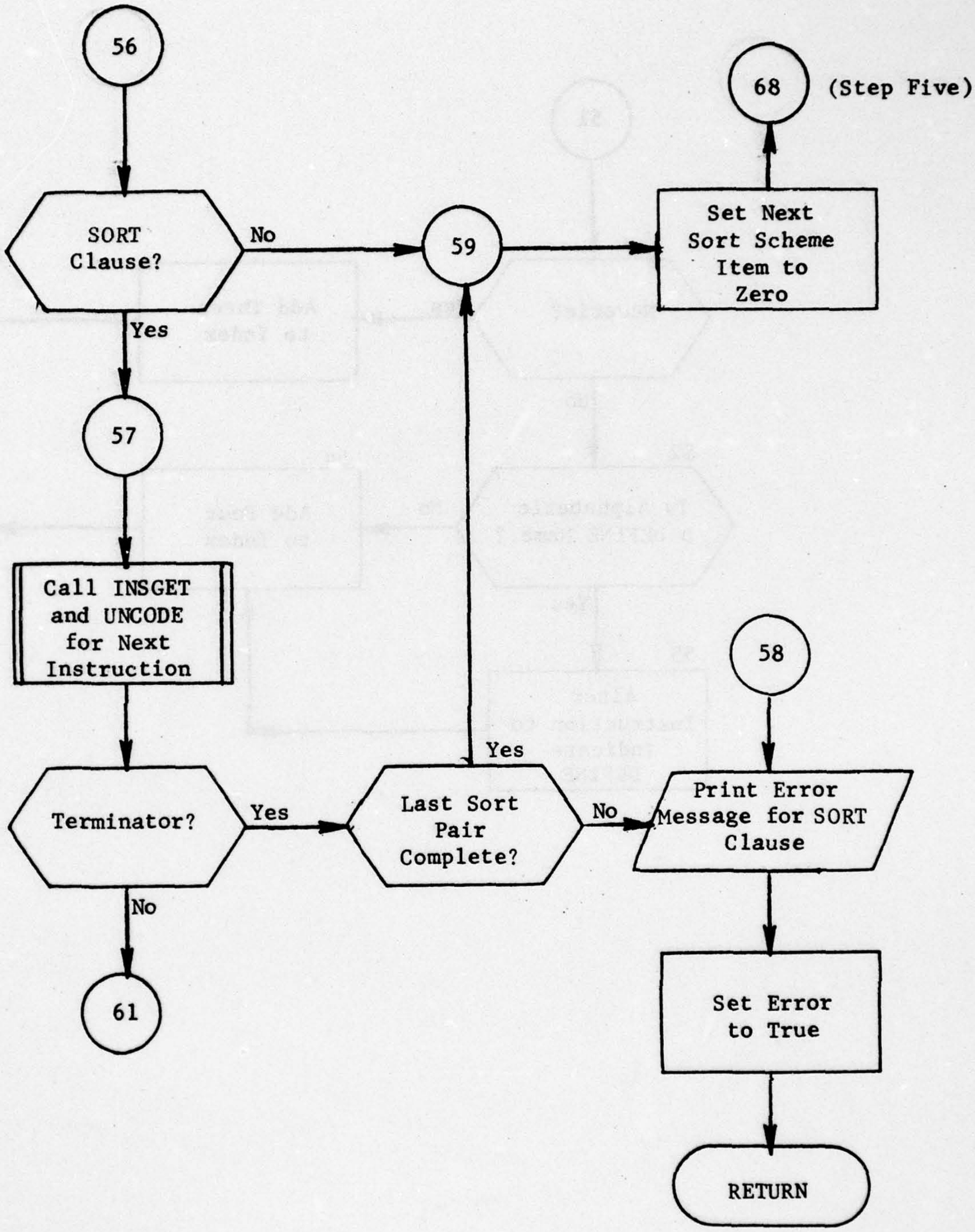
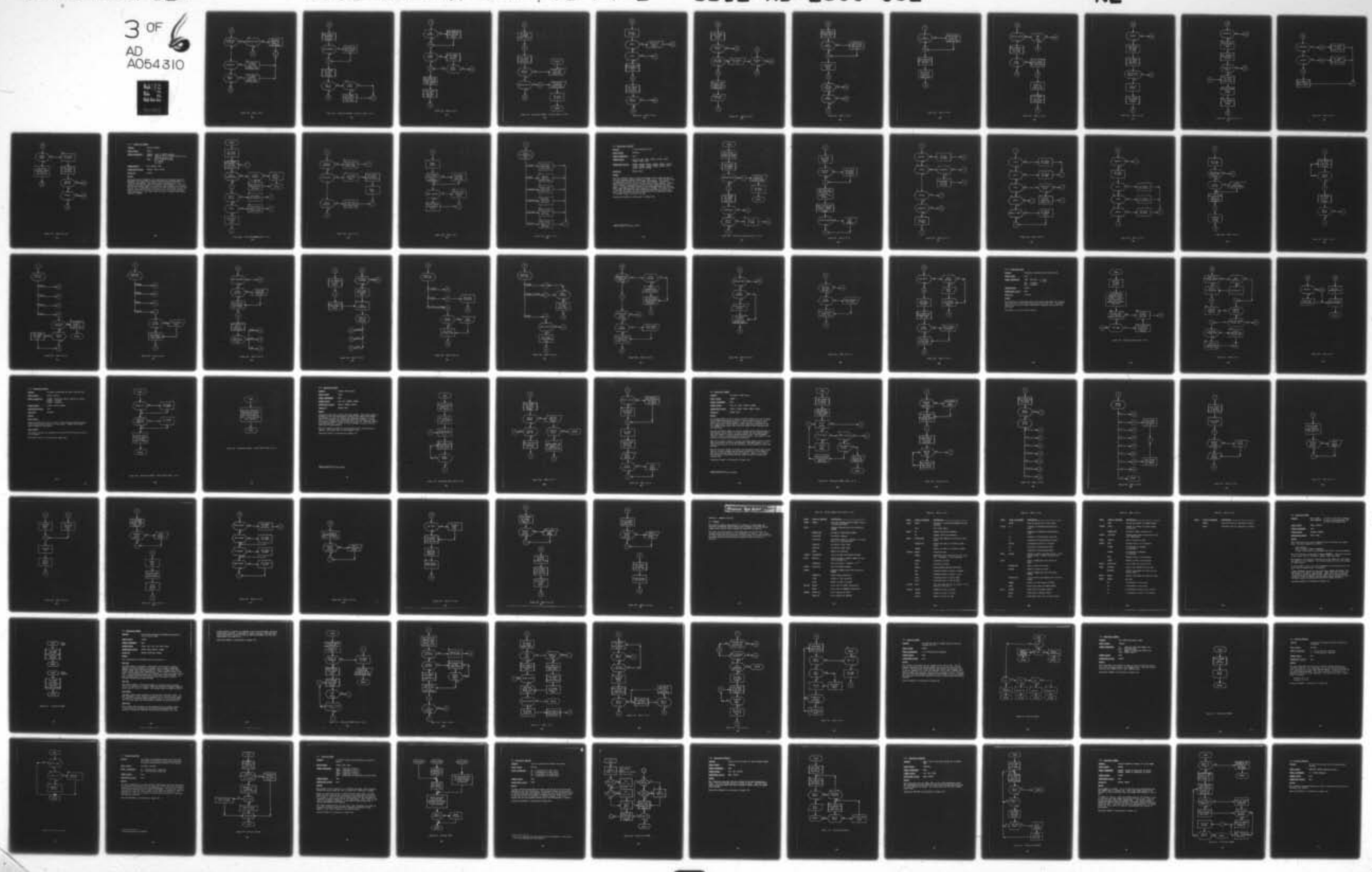


Figure 122. Subroutine BLDOTH: Step Four (Part 1 of 2)

AD-A054 310 COMMAND AND CONTROL TECHNICAL CENTER WASHINGTON D C F/G 15/7
THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK) PROGR--ETC(U)
JUN 77 D J SANDERS, P F MAYKRANTZ, J M HERRIN
UNCLASSIFIED CCTC-CSM-MM-9-77-V1-PT-2 SBIE-AD-E100 052 NL

3 OF 6
AD
A054310



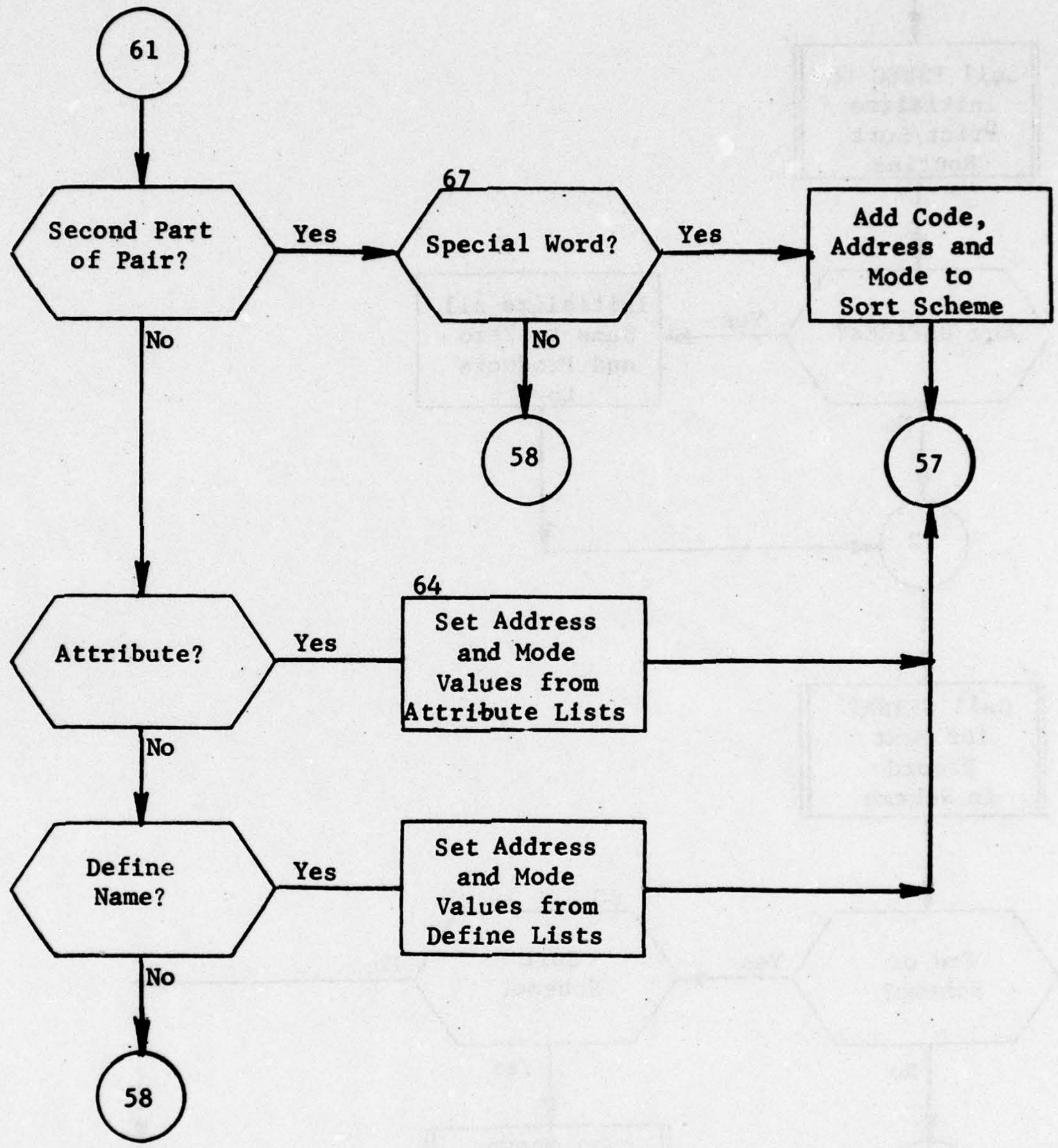


Figure 122. (Part 2 of 2)

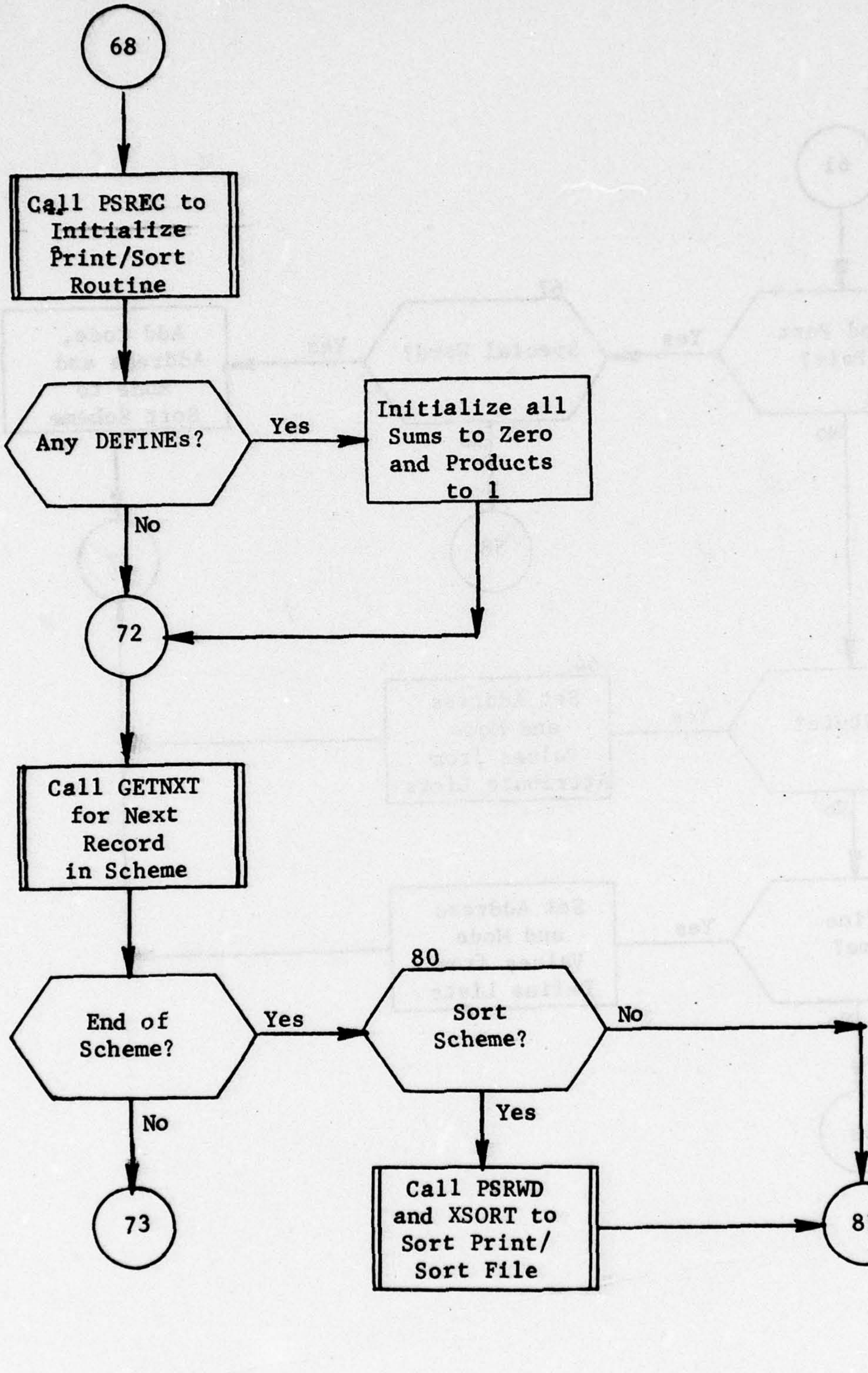


Figure 123. Subroutine BLDOTH: Step Five (Part 1 of 2)

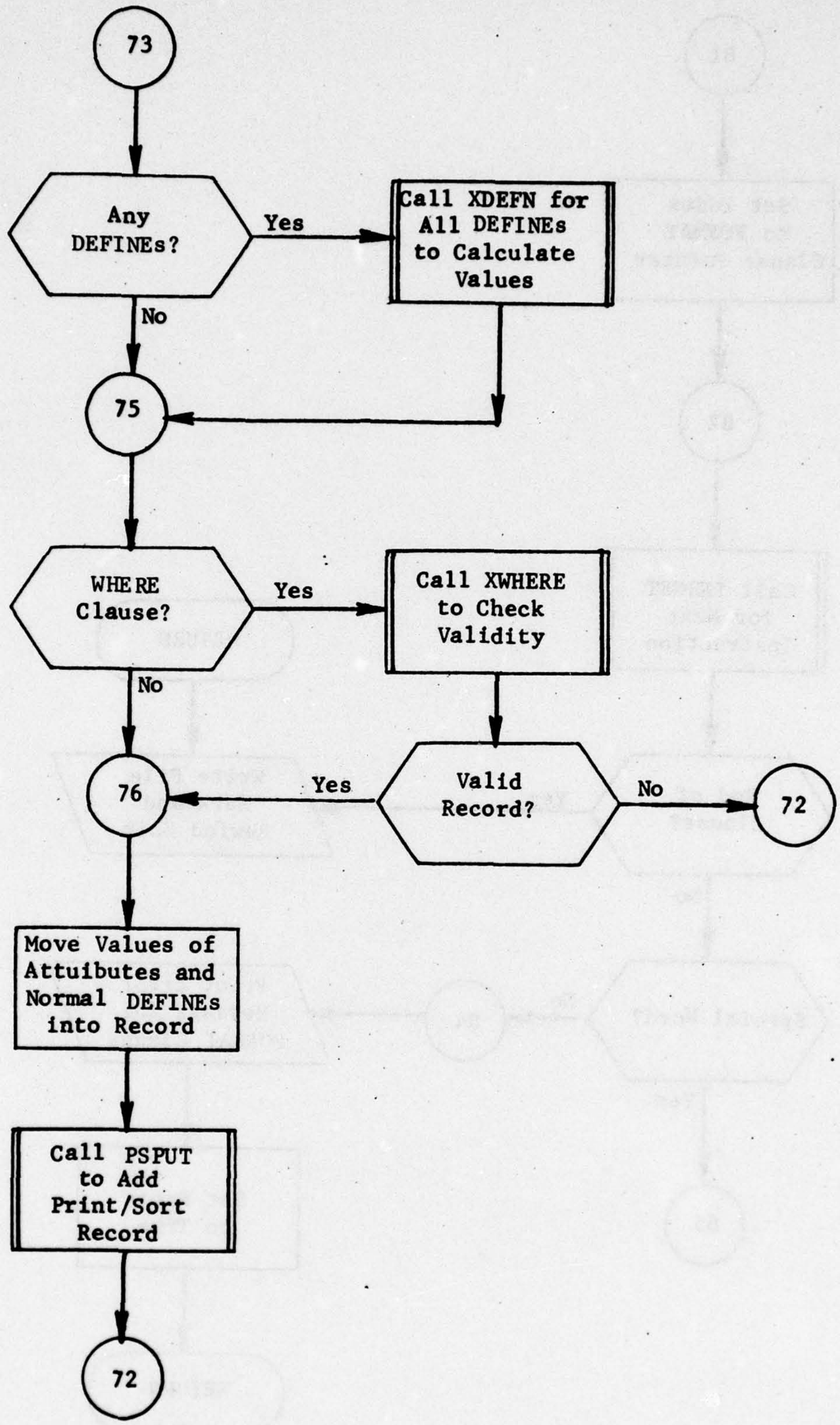


Figure 123. (Part 2 of 2)

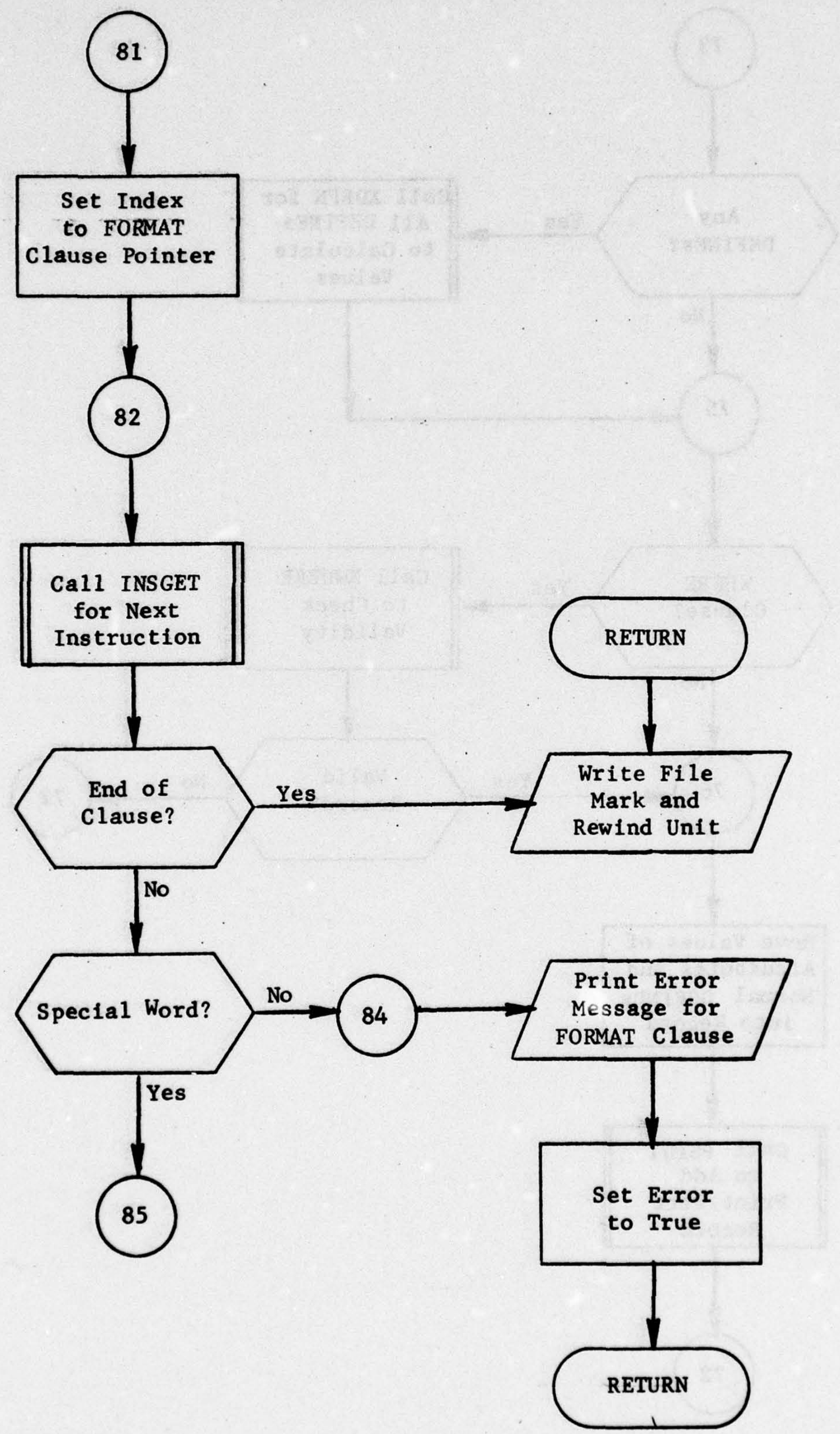


Figure 124. Subroutine BLDOTH: Step Six (Part 1 of 10)

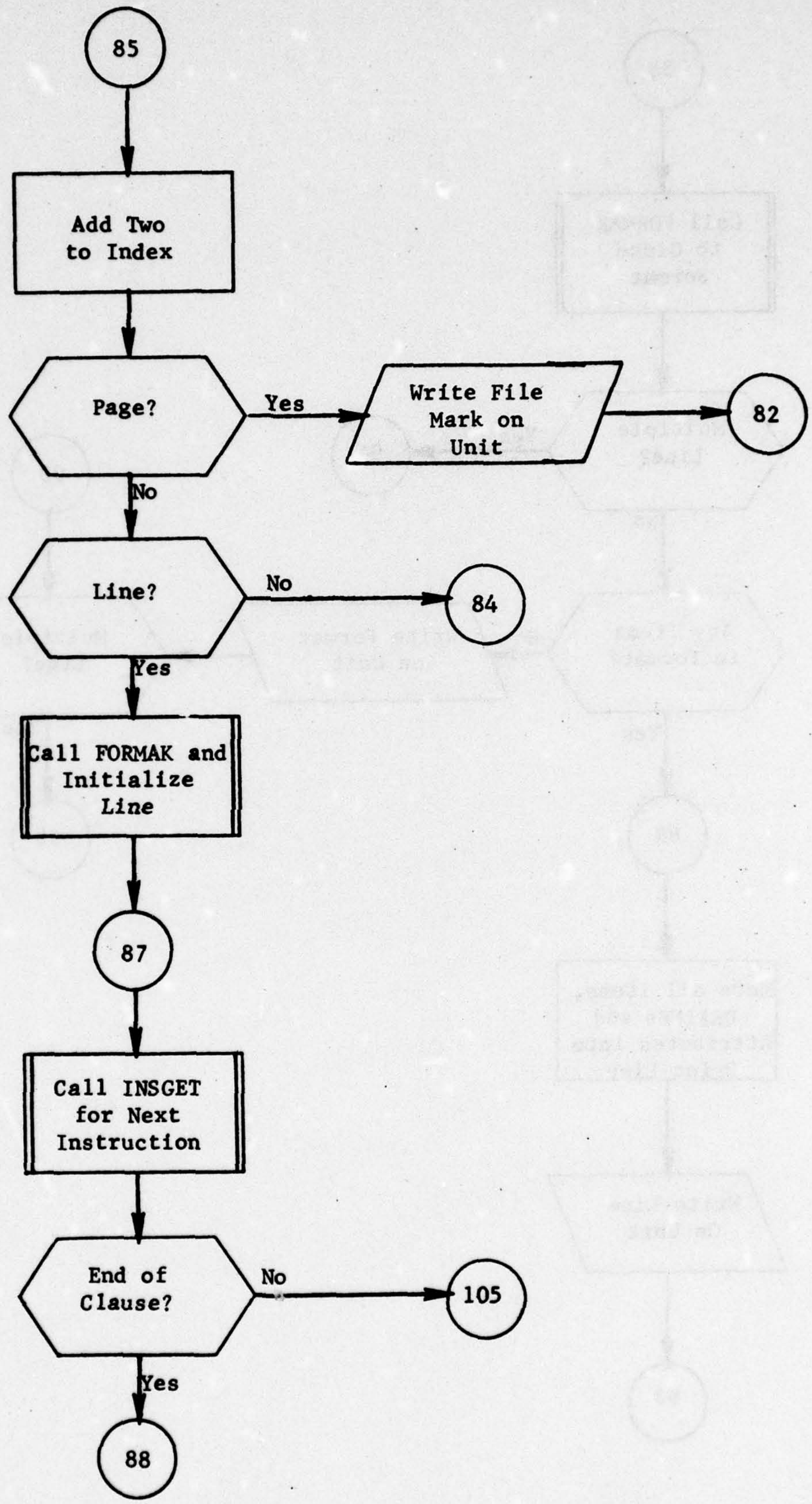


Figure 124. (Part 2 of 10)

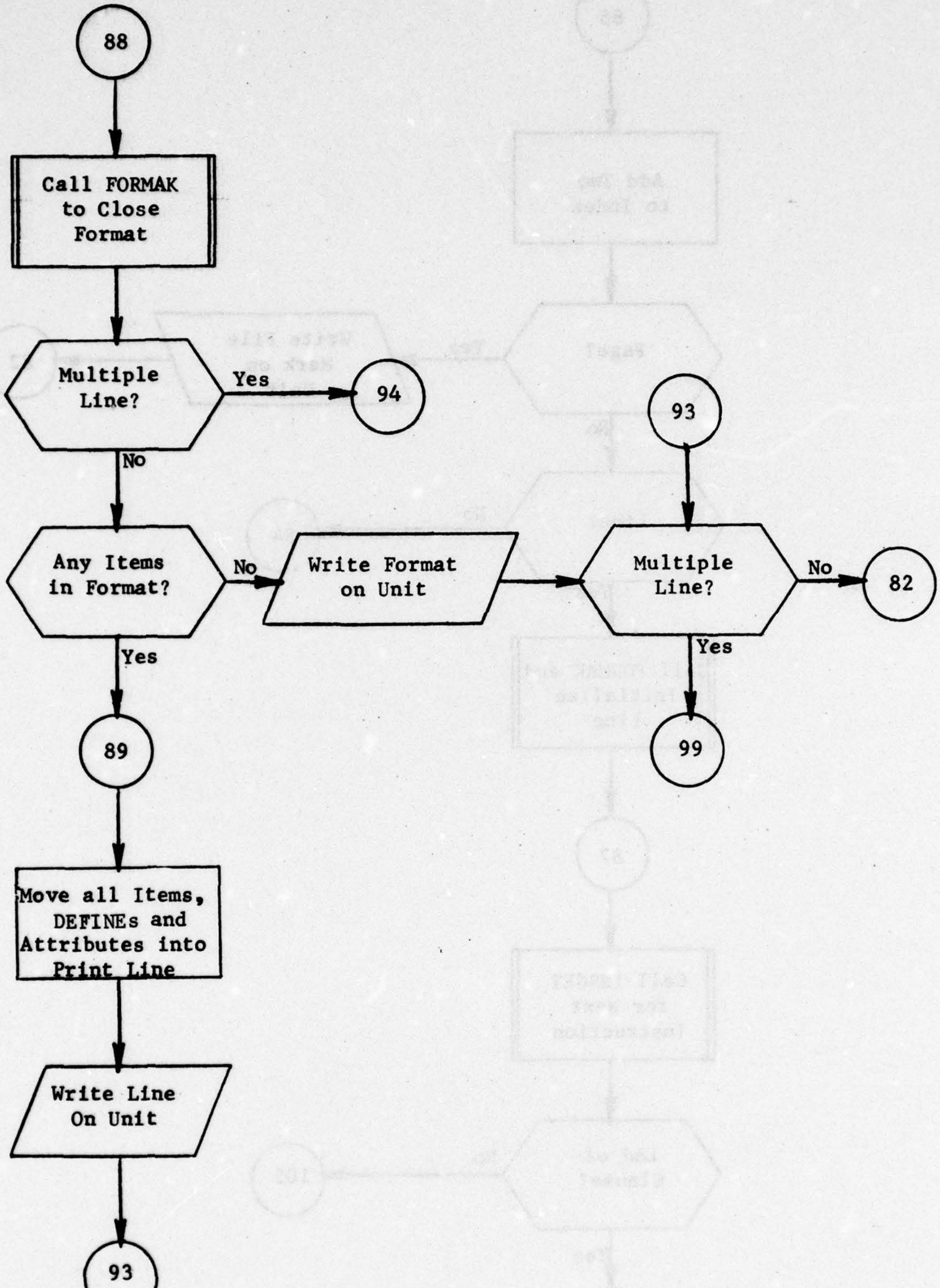


Figure 124. (Part 3 of 10)

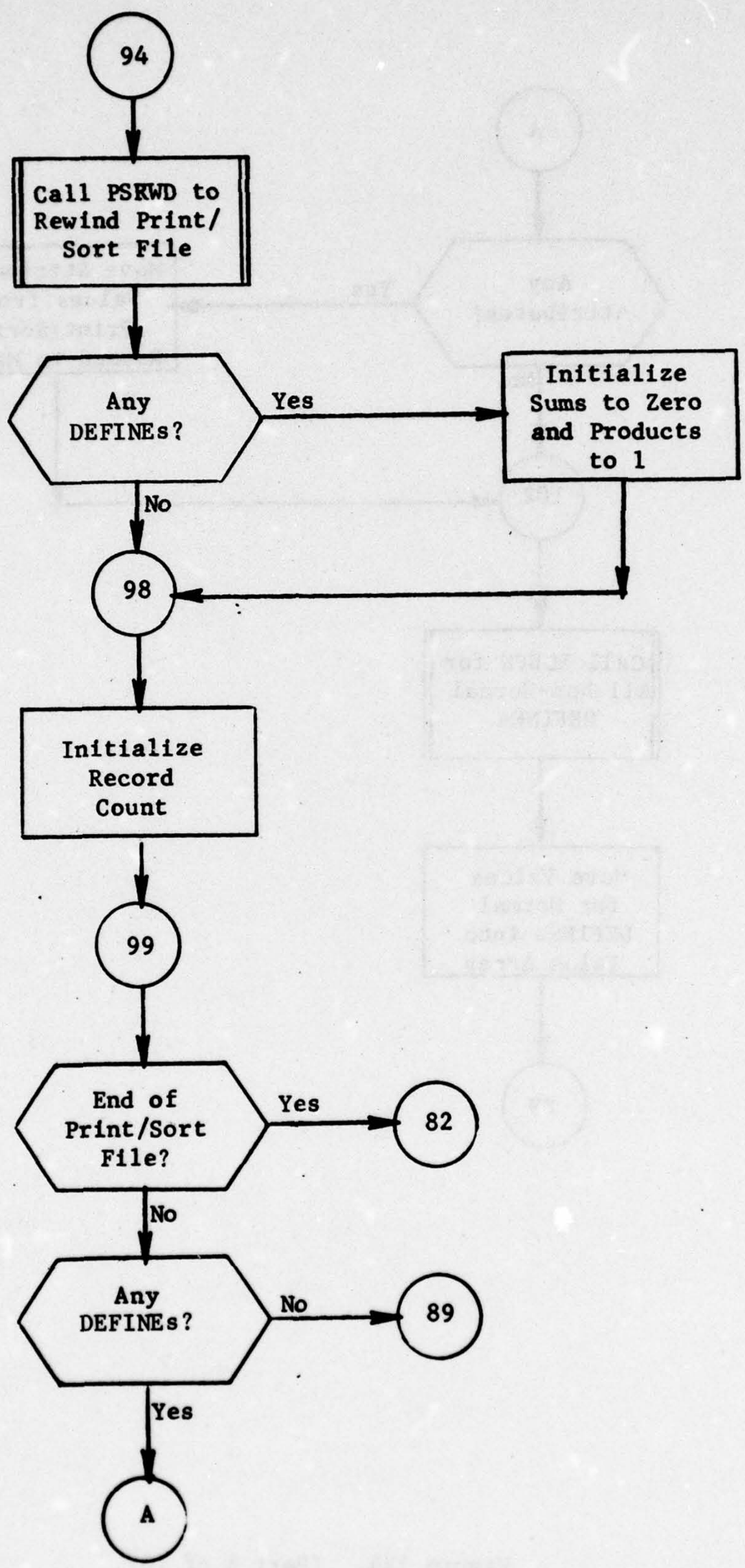


Figure 124. (Part 4 of 10)

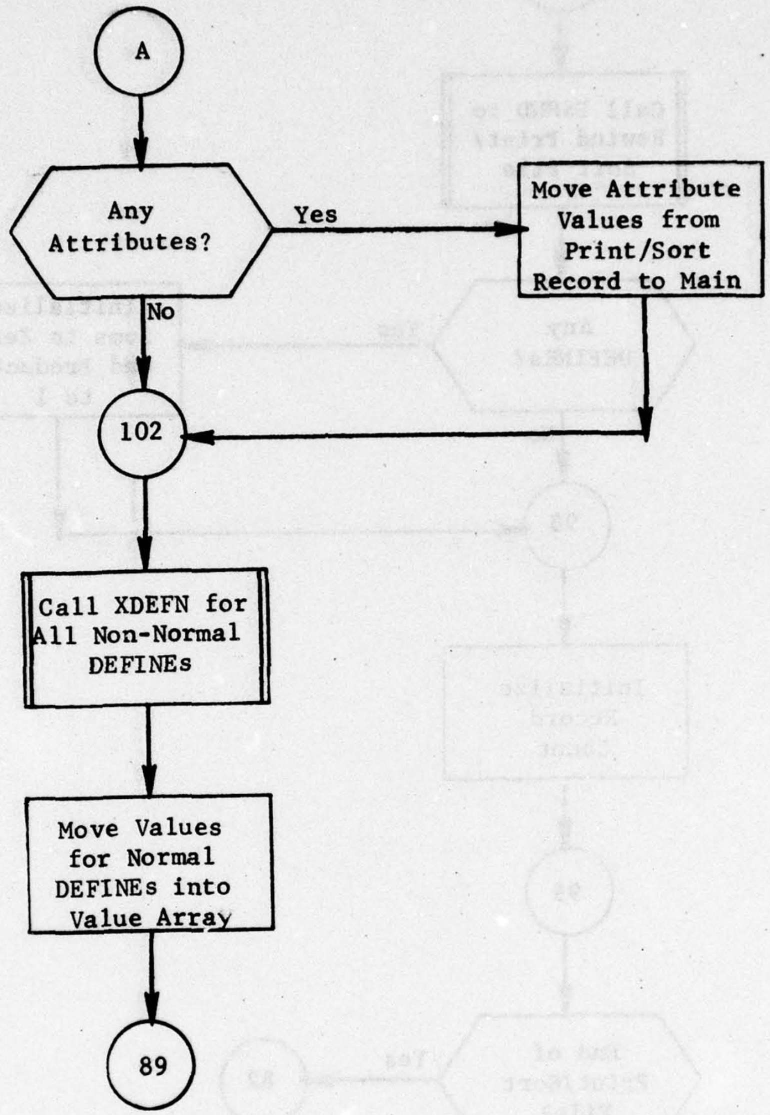


Figure 124. (Part 5 of 10)

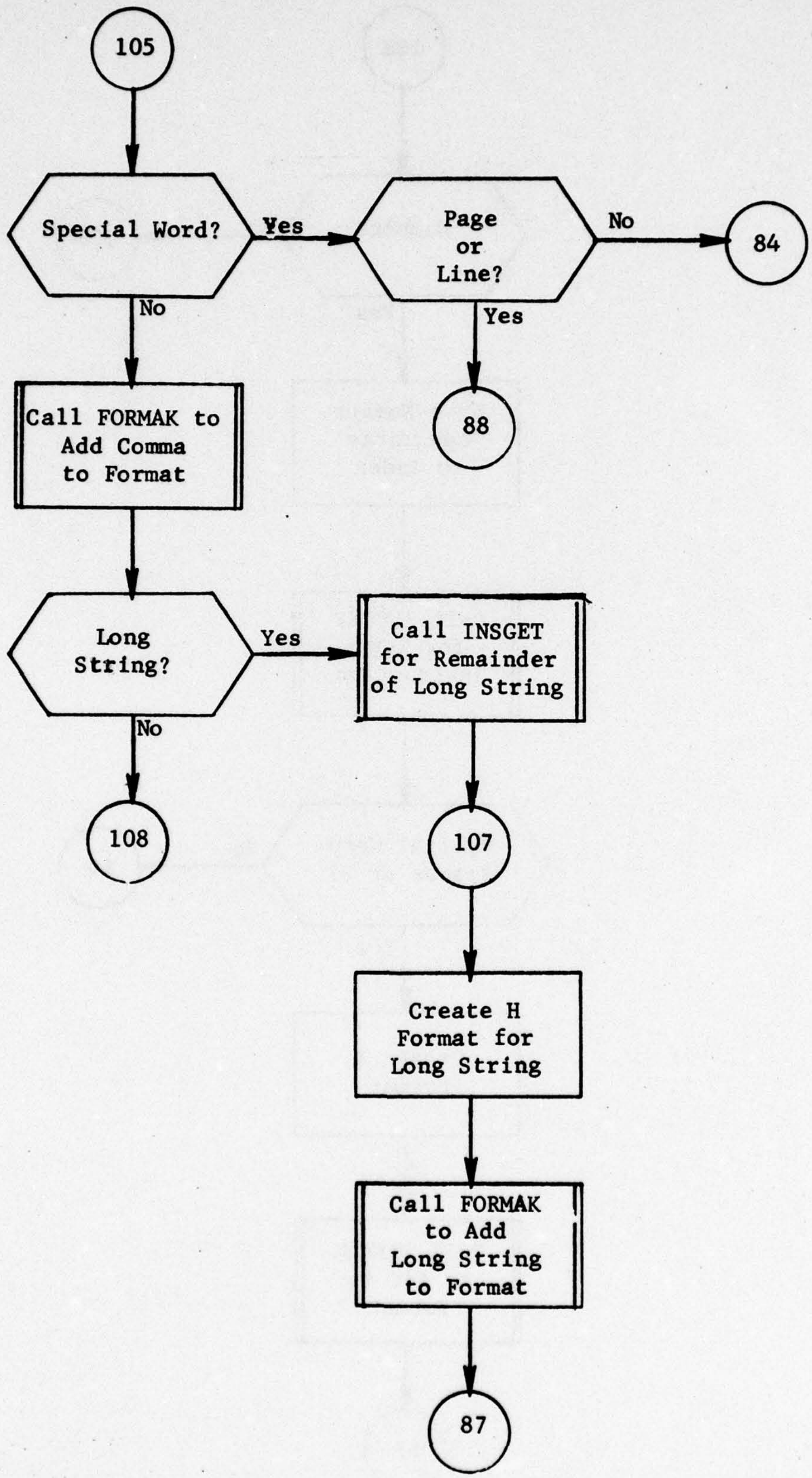


Figure 124. (Part 6 of 10)

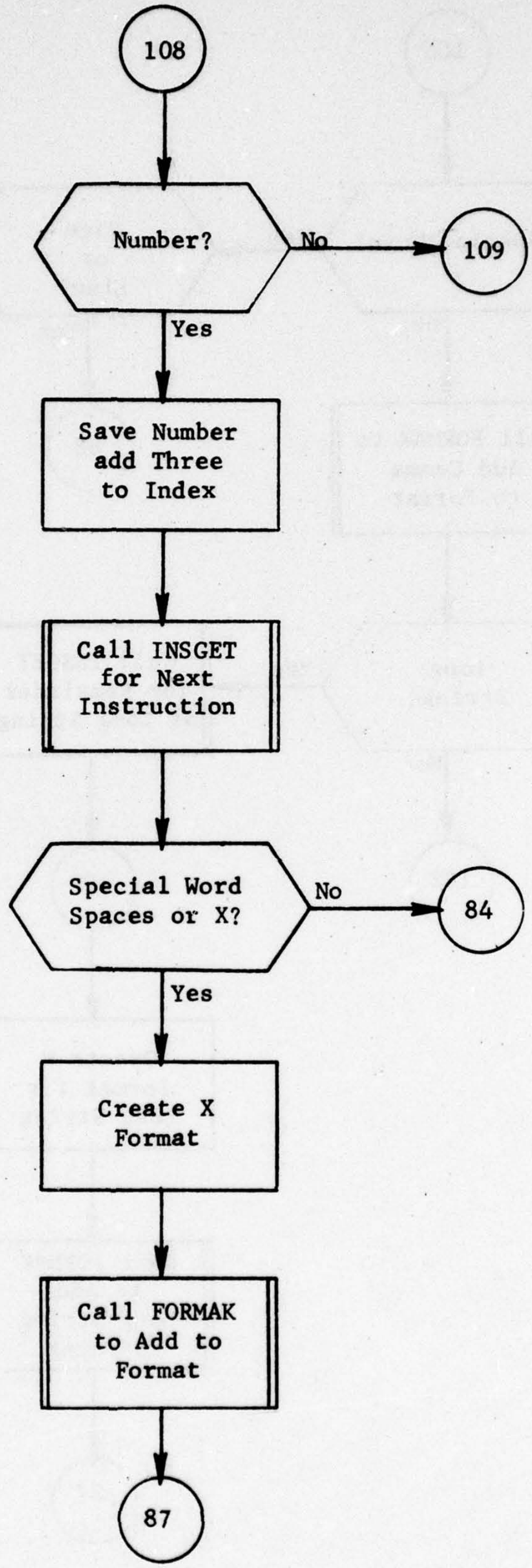


Figure 124. (Part 7 of 10)

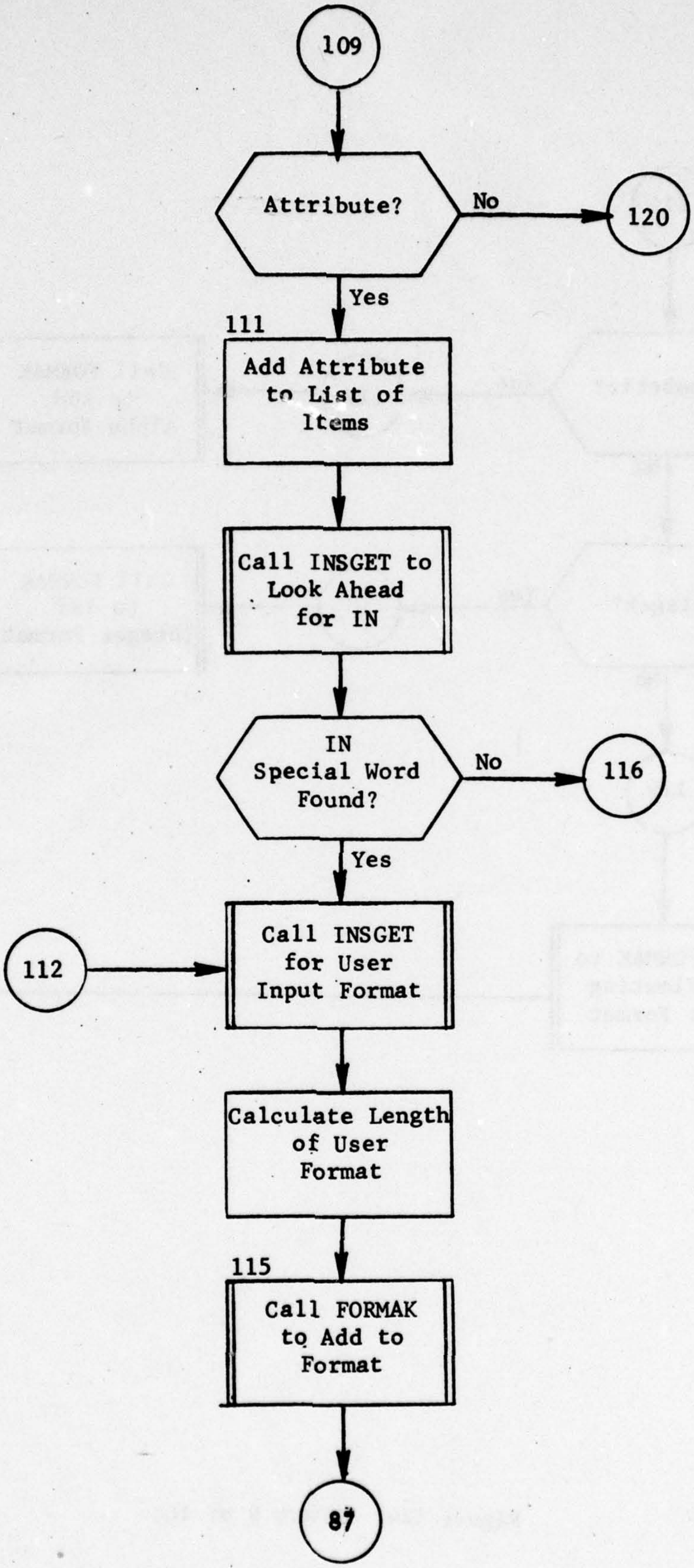


Figure 124. (Part 8 of 10)

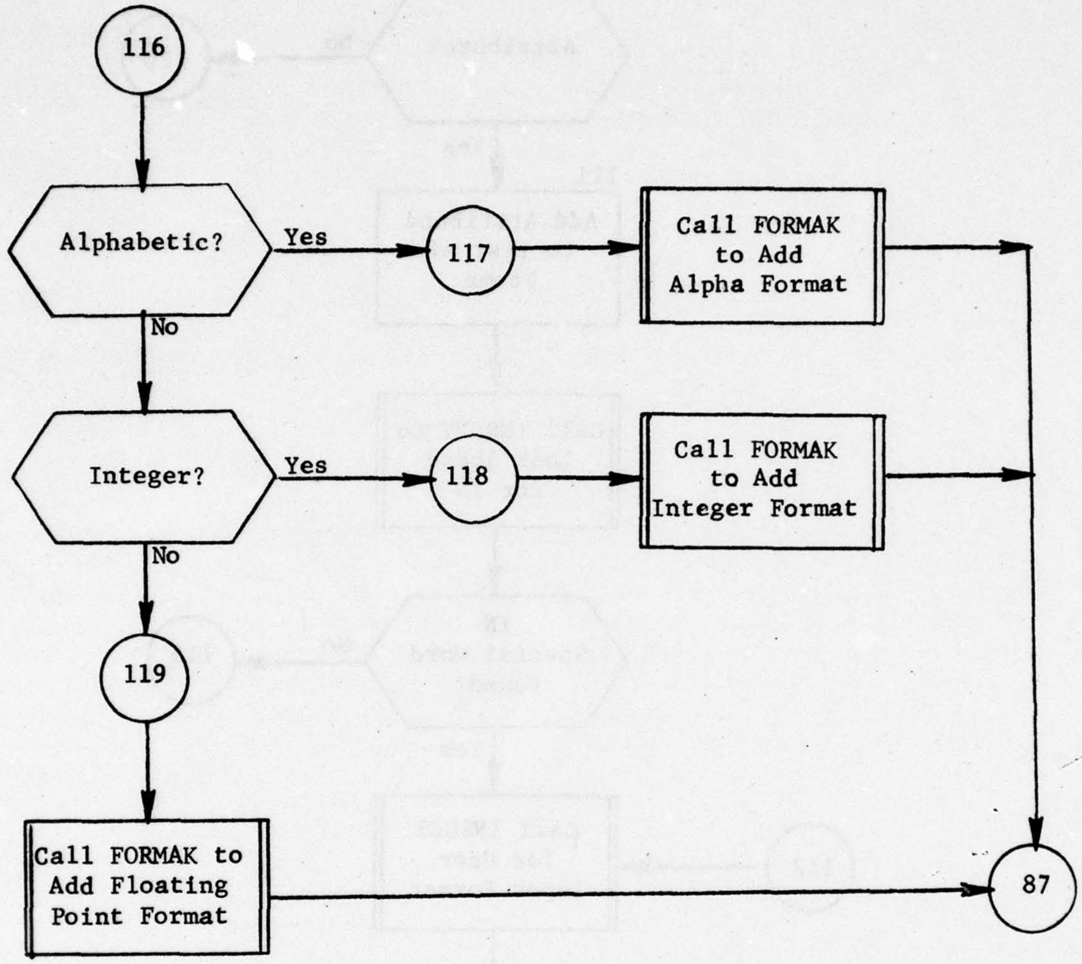


Figure 124. (Part 9 of 10)

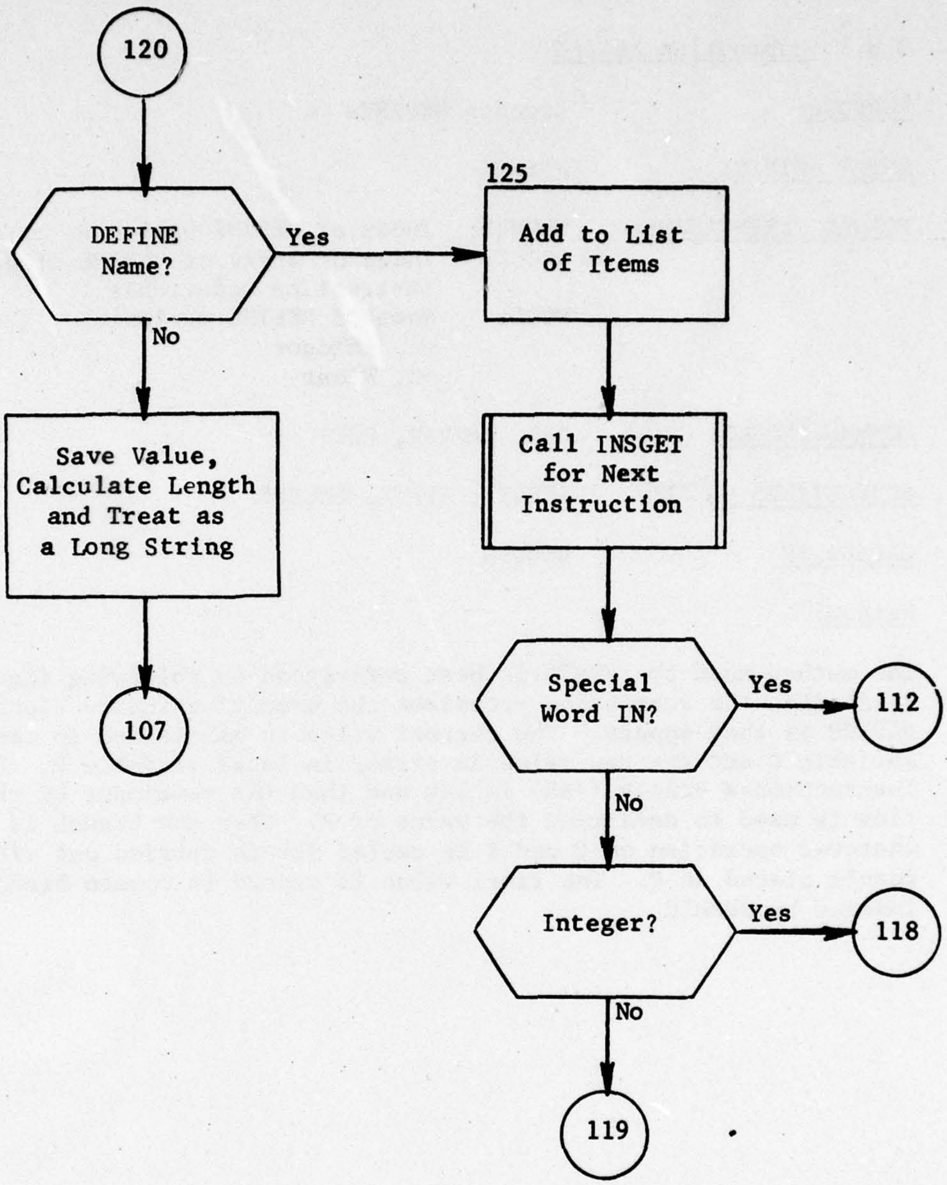


Figure 124. (Part 10 of 10)

8.8.1 Subroutine XEDEFN

PURPOSE: Execute DEFINEs

ENTRY POINTS: XDEFN

FORMAL PARAMETERS:

NUMBER: Index of DEFINE variable
INDEX: Index of start of DEFINE clause in the instruction code table
MODE: Mode of DEFINE variable
=1, Integer
=2, Float

COMMON BLOCKS: C30, DEFVAR, ZEES

SUBROUTINES CALLED: INSGET, IORFL, UNCODE

CALLED BY: BLDOTH

Method:

The method used by XEDEFN is best understood by following figure 125. Basically, the subroutine processes the execution instructions for a DEFINE as they appear. The current value is maintained in local variable Q and the new value is stored in local variable R. For each instruction a branch (IBR) is set and then the remainder of the instruction is used to determine the value of R. Then the branch is made and whatever operation on Q and R is called for is carried out with the result placed in Q. The final value is stored in common block DEFVAR indexed by NUMBER.

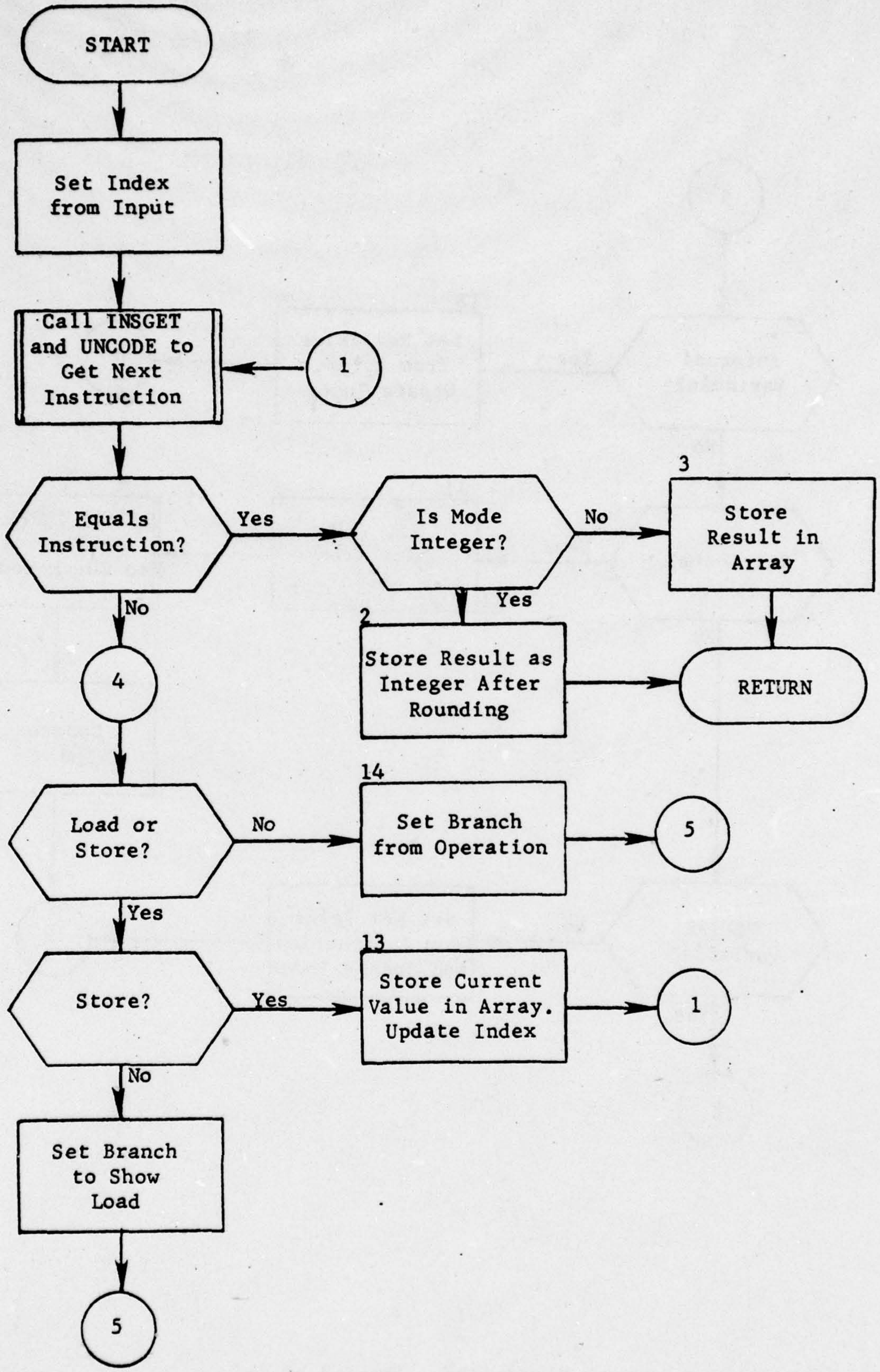


Figure 125. Subroutine XEDEFN (Part 1 of 4)

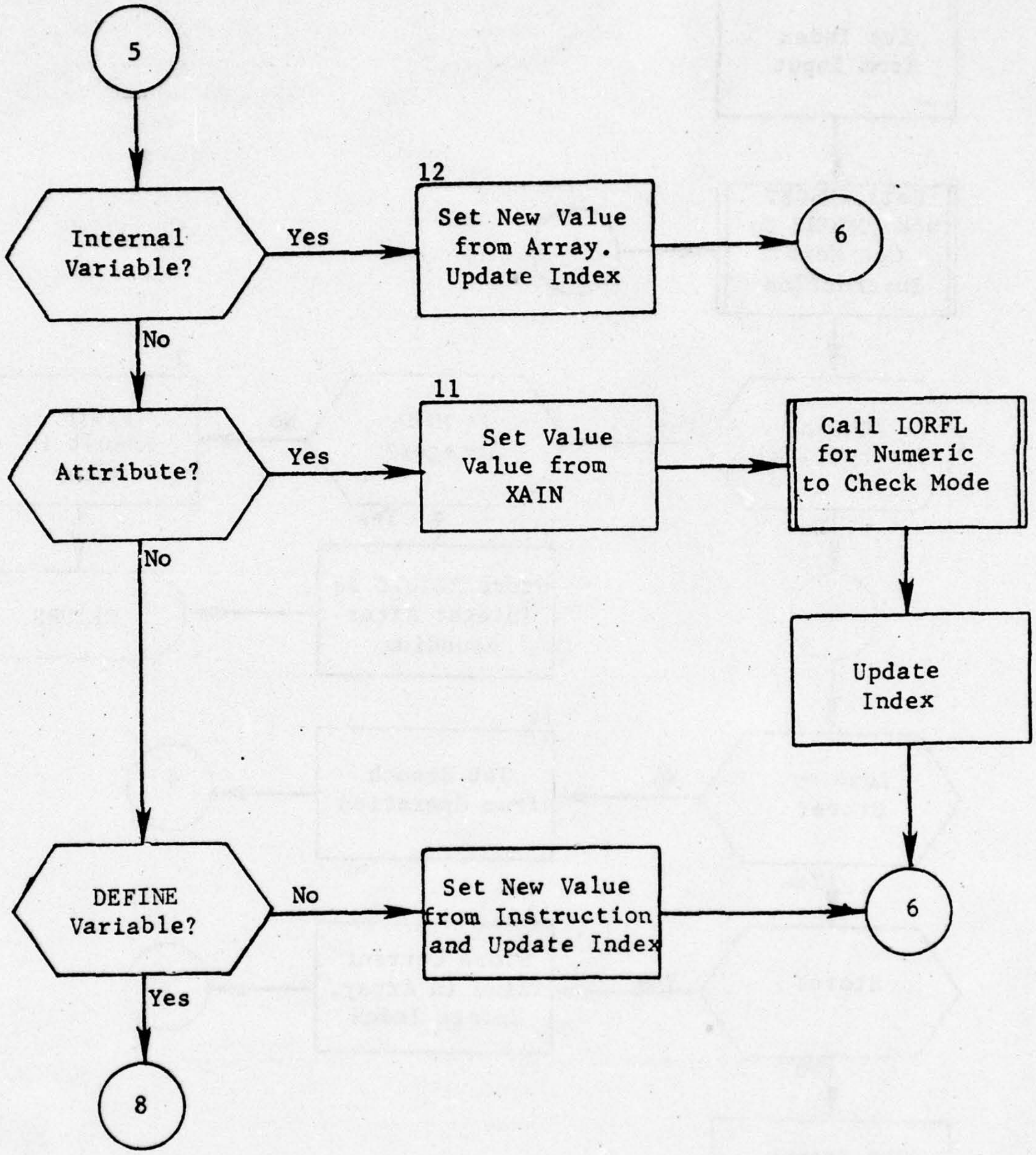


Figure 125. (Part 2 of 4)

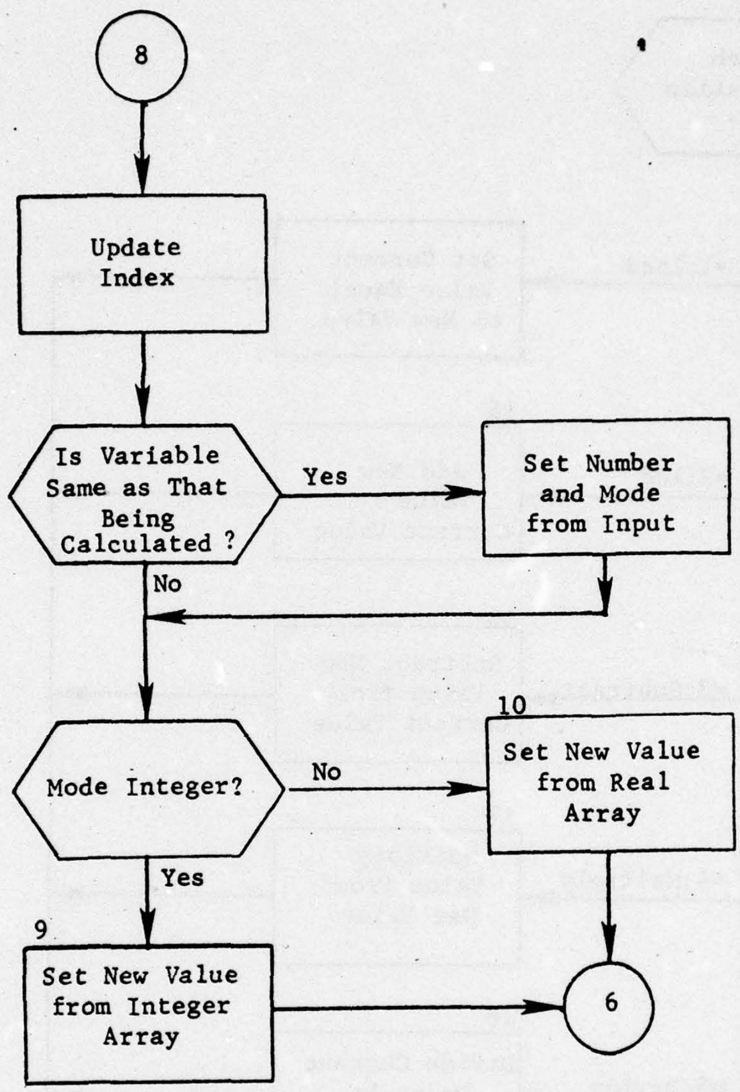


Figure 125. (Part 3 of 4)

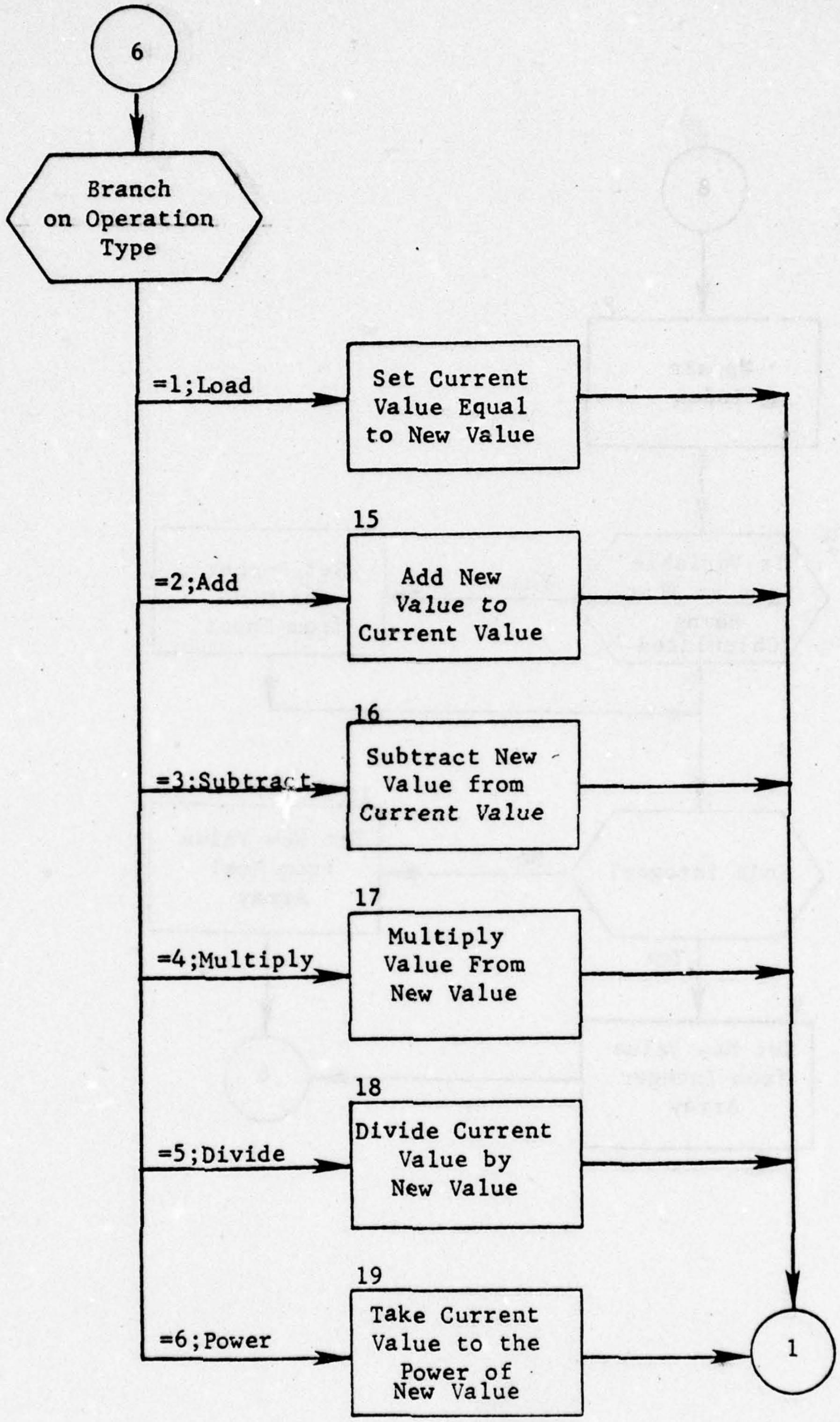


Figure 125. (Part 4 of 4)

8.9 Subroutine PLOTDATA*

PURPOSE: To plot geographic data

ENTRY POINTS: PLOTDATA

FORMAL PARAMETERS: None

COMMON BLOCKS: C15, C20, C30, OOPS, PLTPRO, PLTSPE, PRWSP, SCHEME, TAPES, ZEES

SUBROUTINES CALLED: GETNXT, HOUSKE, HOUSK2, INSGET, NEWPEV, NEXTTT, PIECEI, PIECEN, PIECE1, PIECE3, PIECE4, PICS, PROJCT, PROJT2, UNCODE

CALLED BY: ENTMOD (EIM)

Method:

First the SETTING clause is read for changes to the normal settings of the plot selection factors (SCALE, LAT, LONG, etc.). Next the plot tape is initialized through calls to PIECEIT and HOUSKEEP. The general method from then on is for each of the types of plots desired: penetration corridors, depenetration corridors, refuel points and recovery bases. A preset retrieval scheme is read from subroutine data (arrays: SCHA, SCHB, SCHC and SCHD) into common block SCHEME. The value for SIDE is inserted and GETNXT is used to retrieve each desired item for the plot. The coordinates are then adjusted by PICS or PROJT2 and PIECEIT or PIECENUM called to perform the plot.

Subroutine PLOTDATA is illustrated in figure 126.

*Main routine of overlay PLOTIT

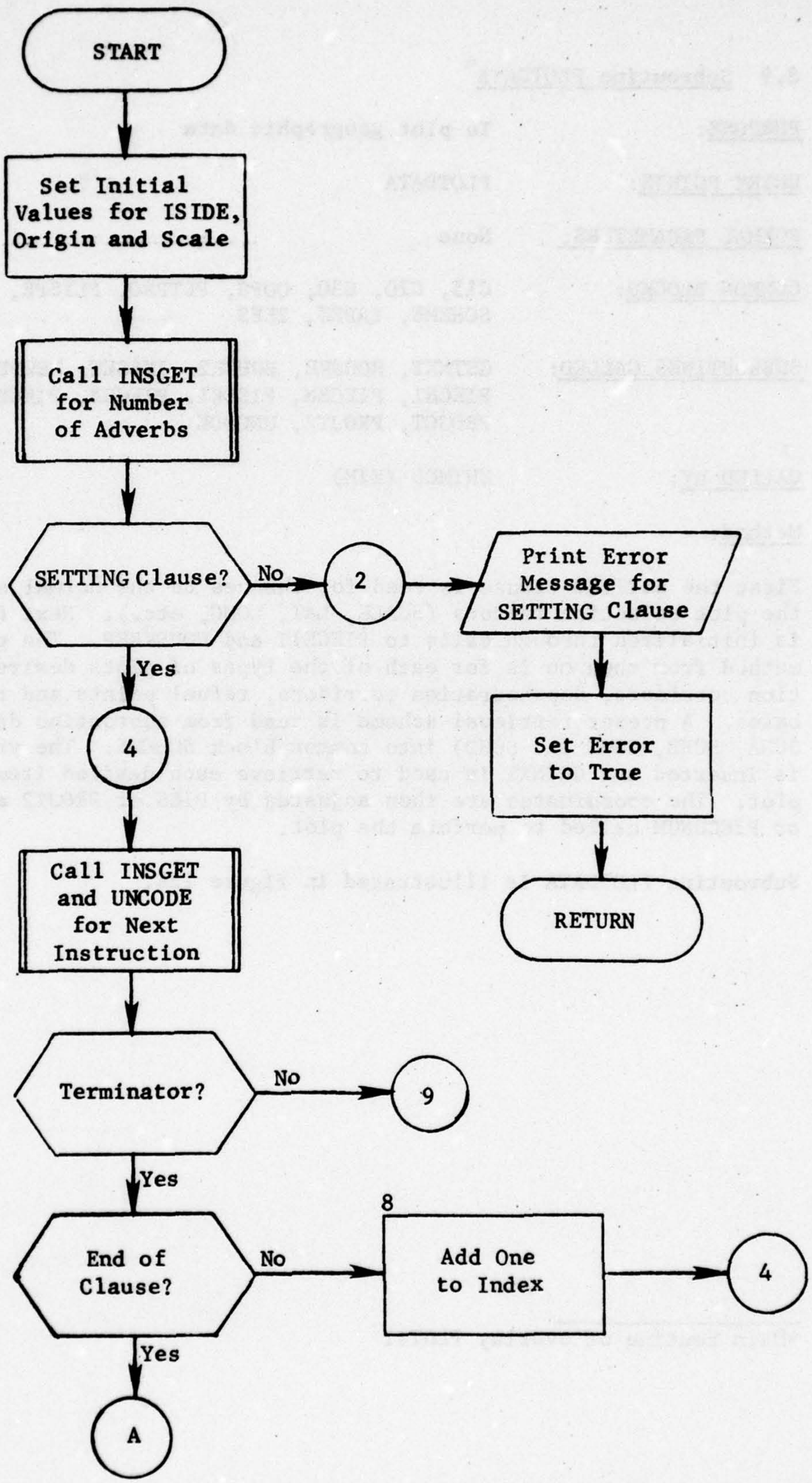


Figure 126. Subroutine PLOTDATA (Part 1 of 17)

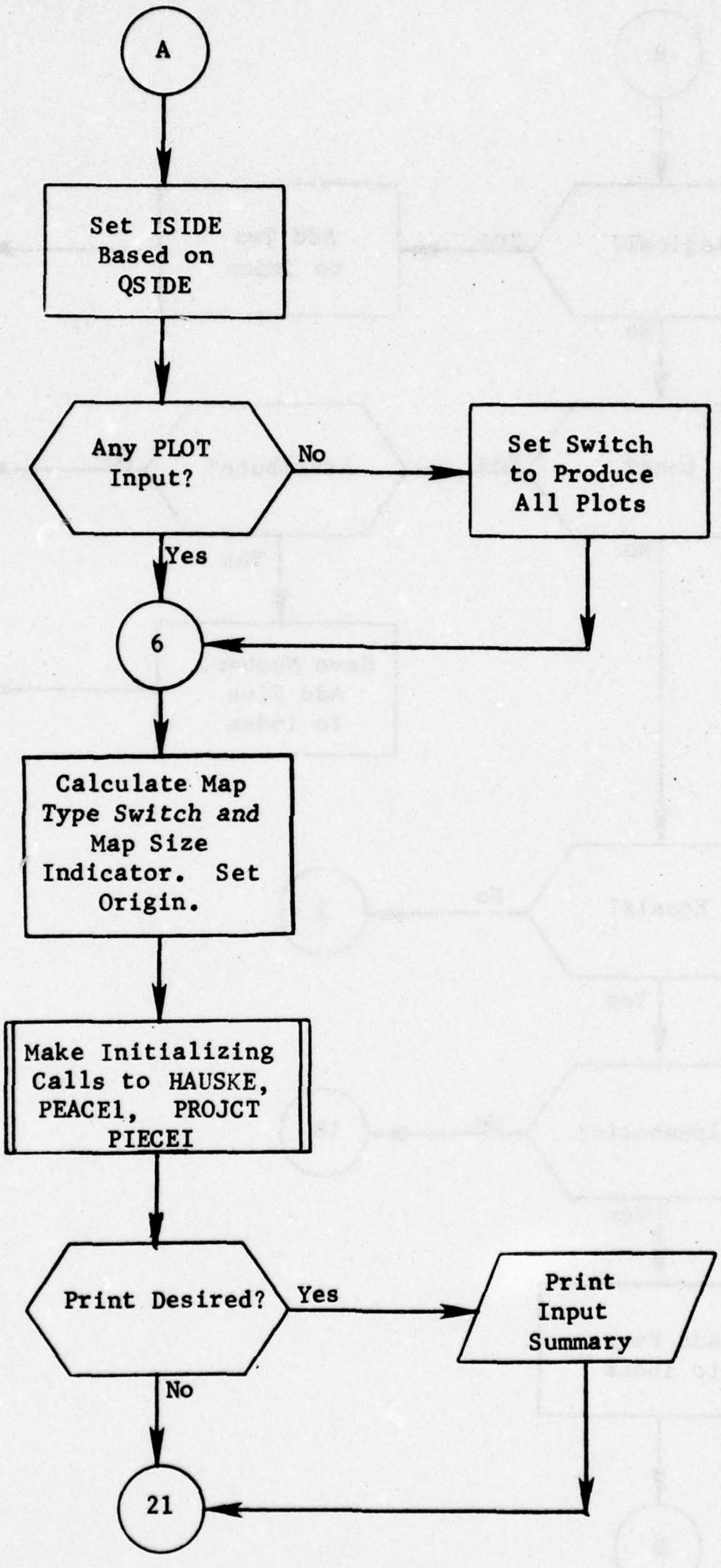


Figure 126. (Part 2 of 17)

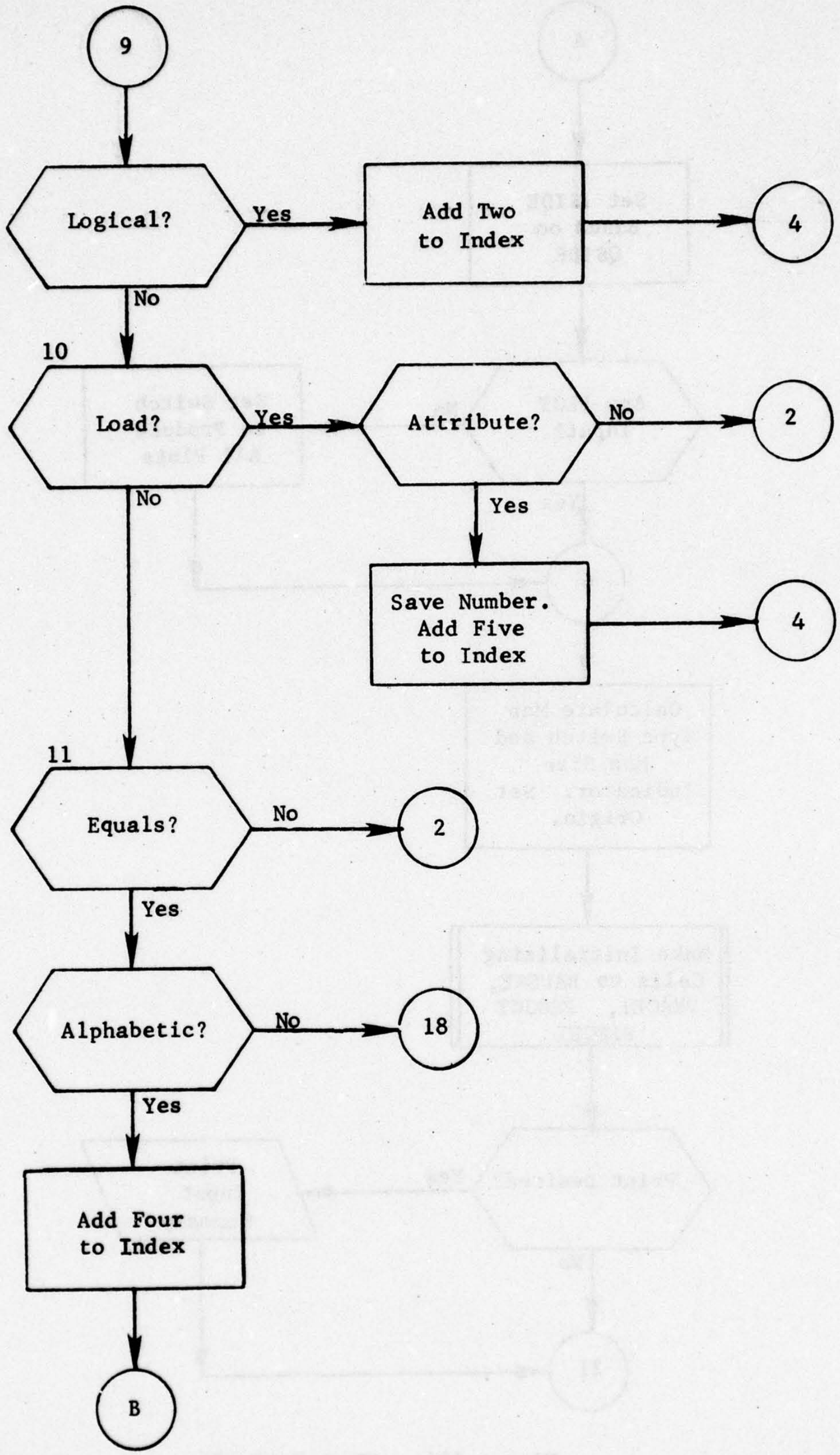


Figure 126. (Part 3 of 17)

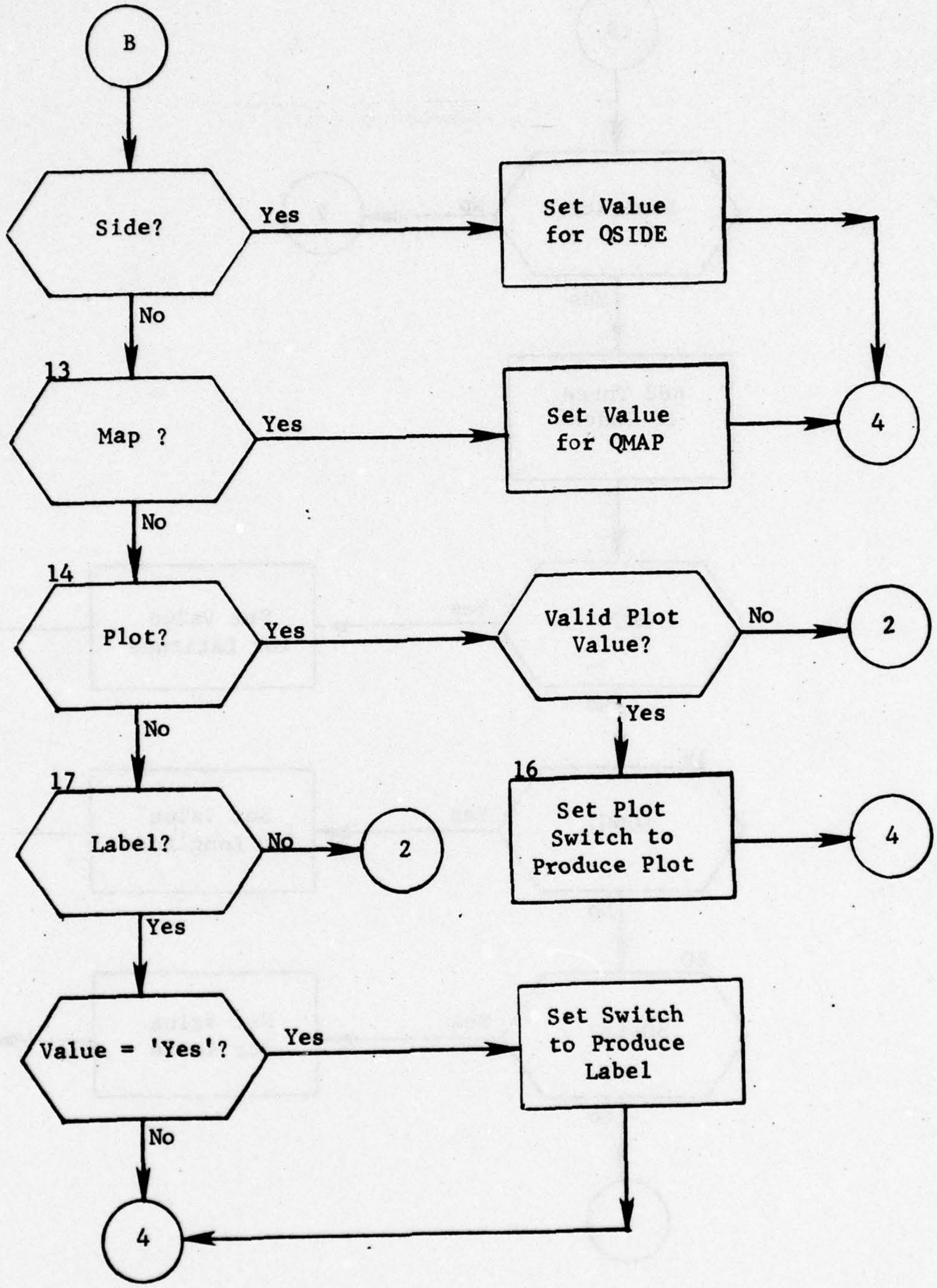


Figure 126. (Part 4 of 17)

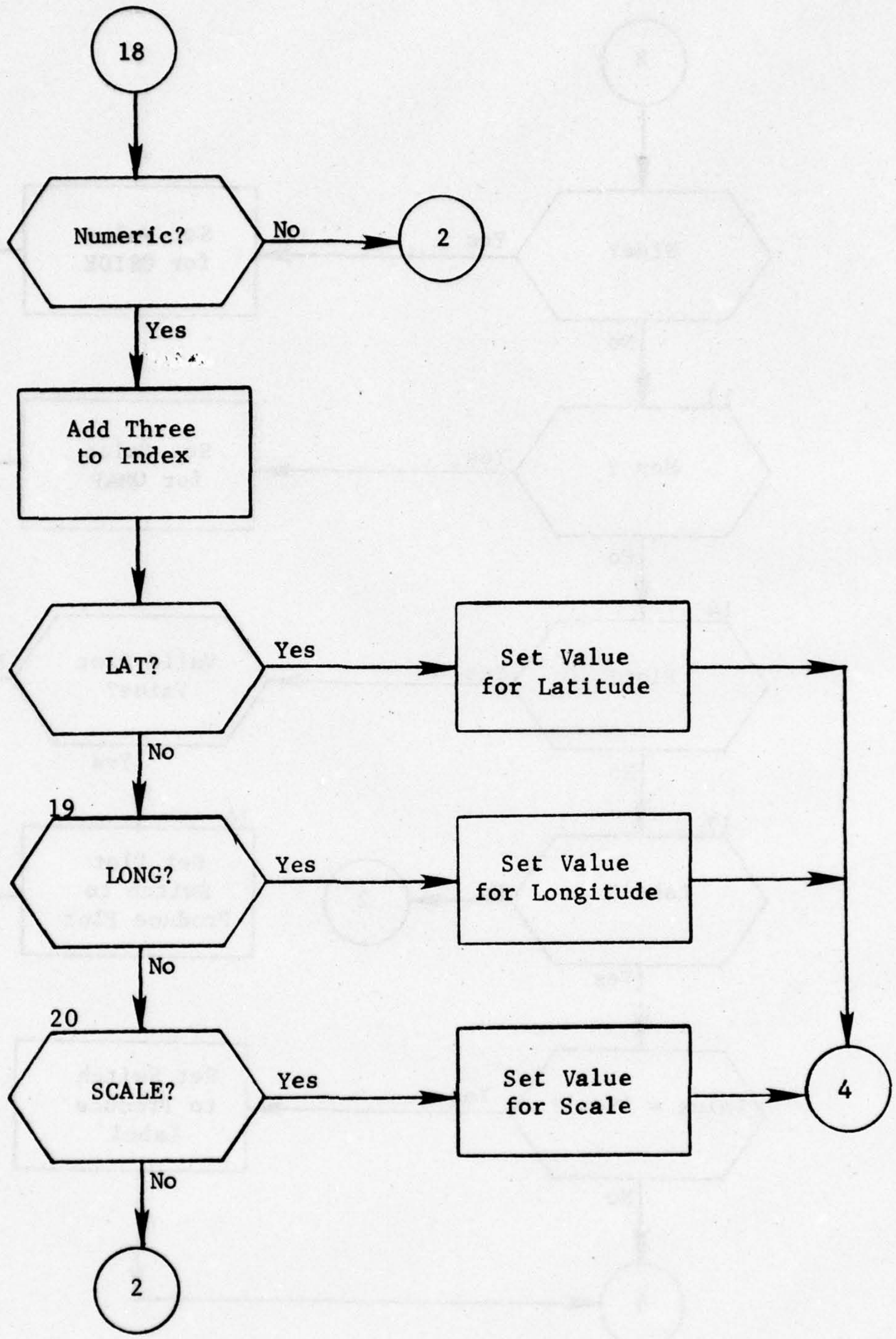


Figure 126. (Part 5 of 17)

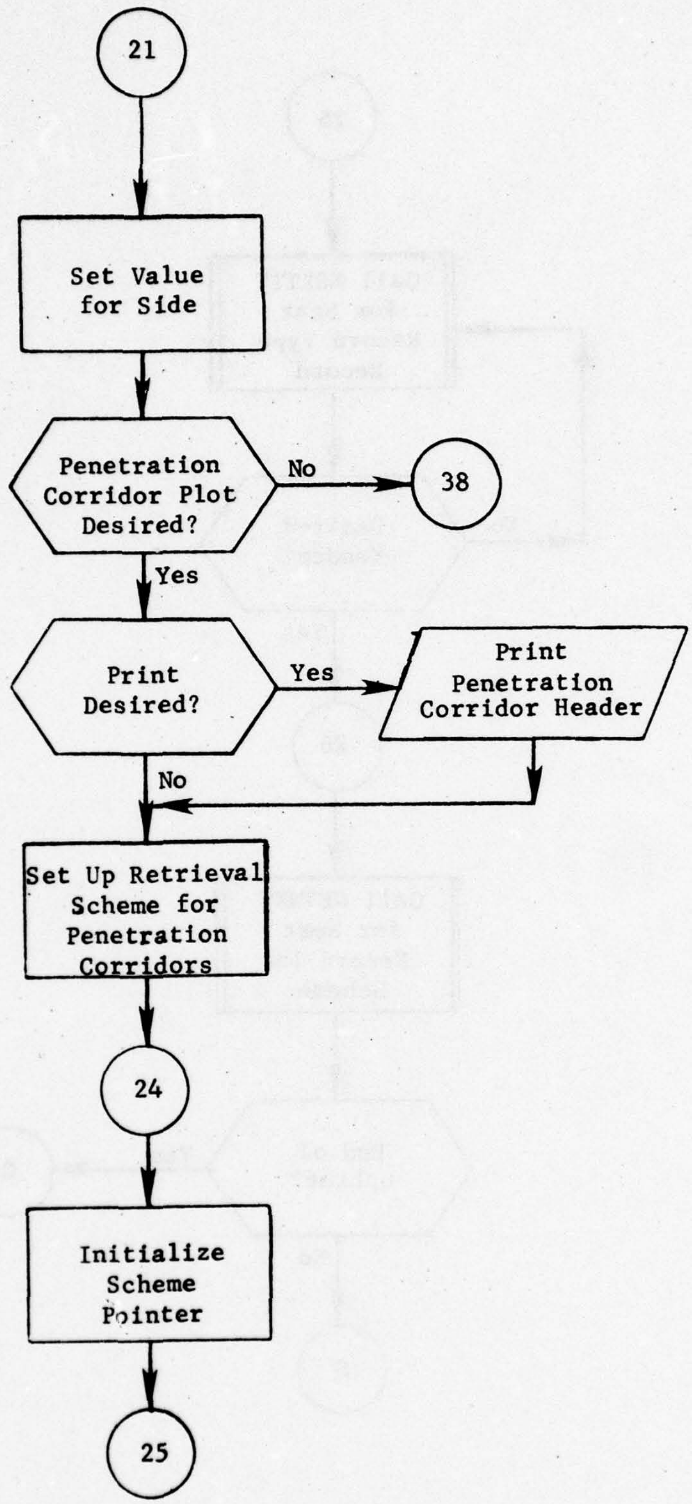


Figure 126. (Part 6 of 17)

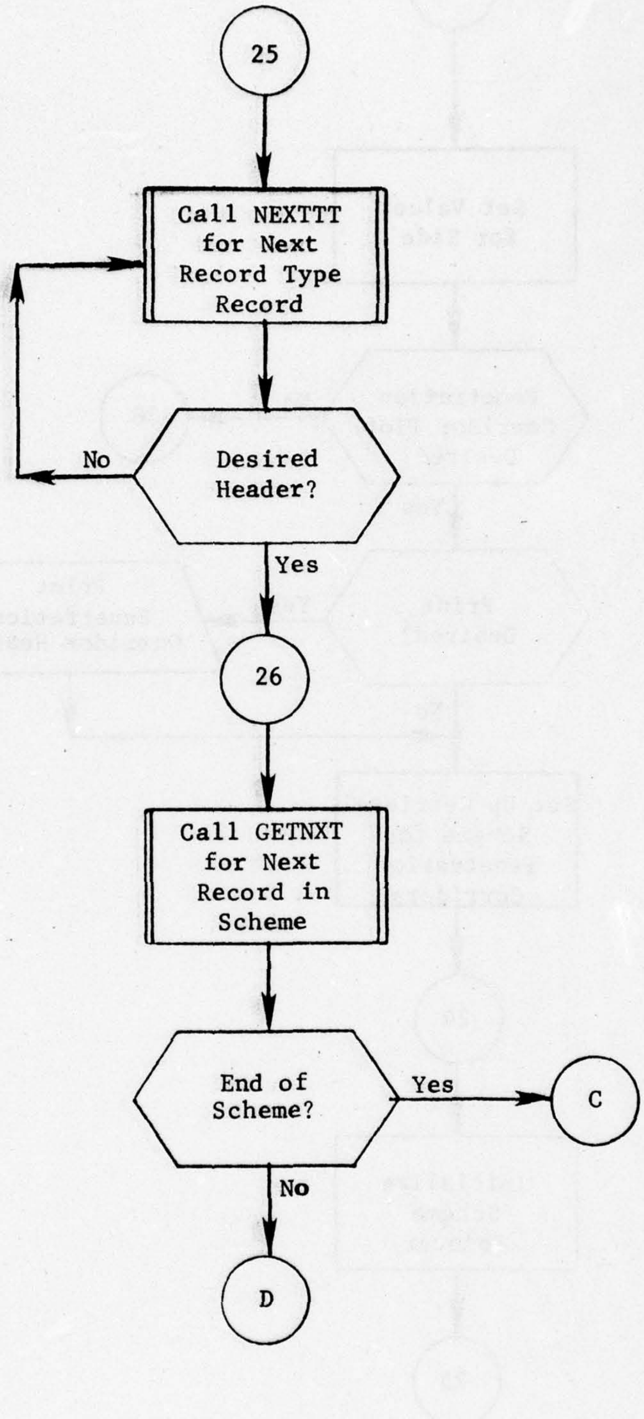


Figure 126. (Part 7 of 17)

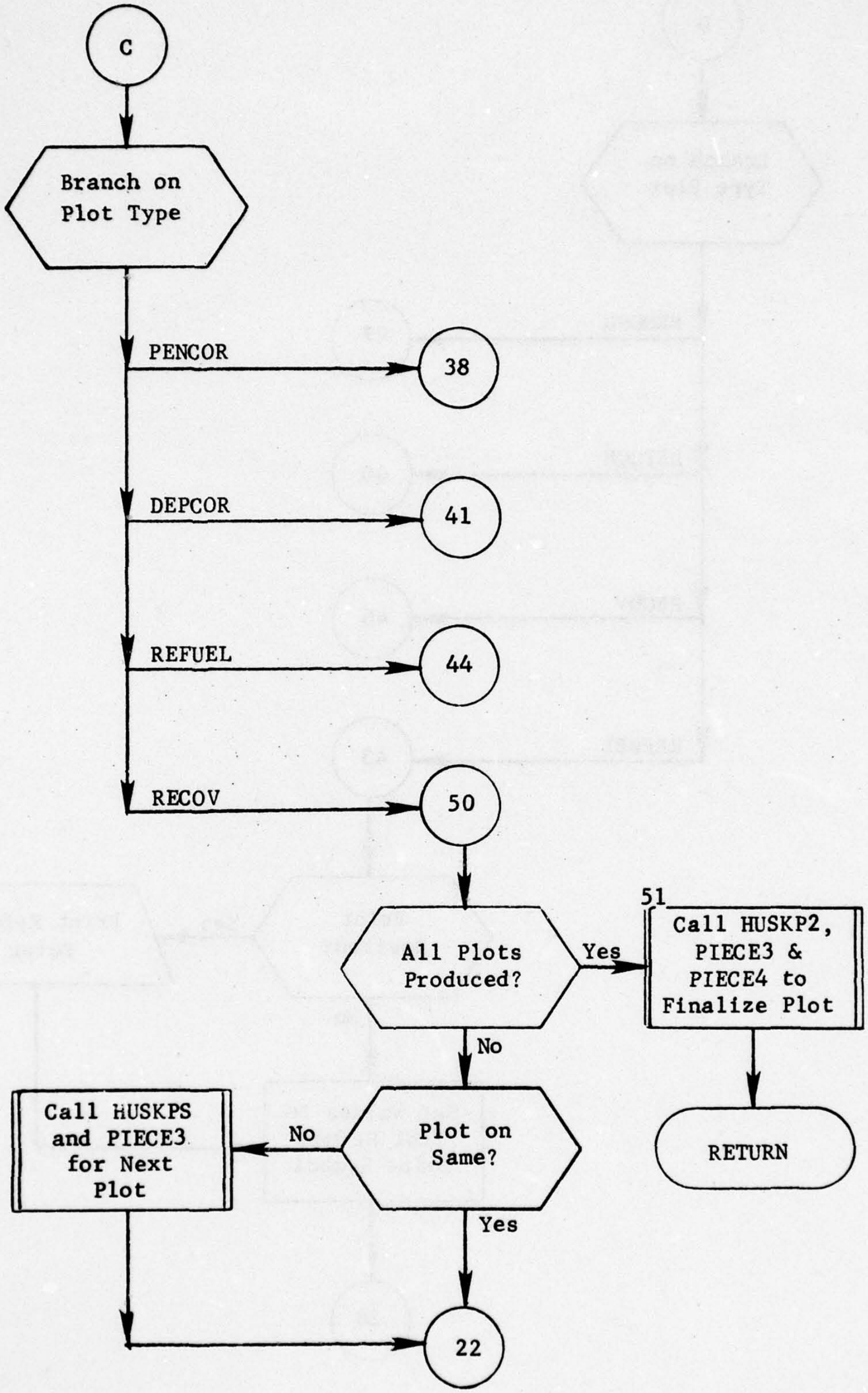


Figure 126. (Part 8 of 17)

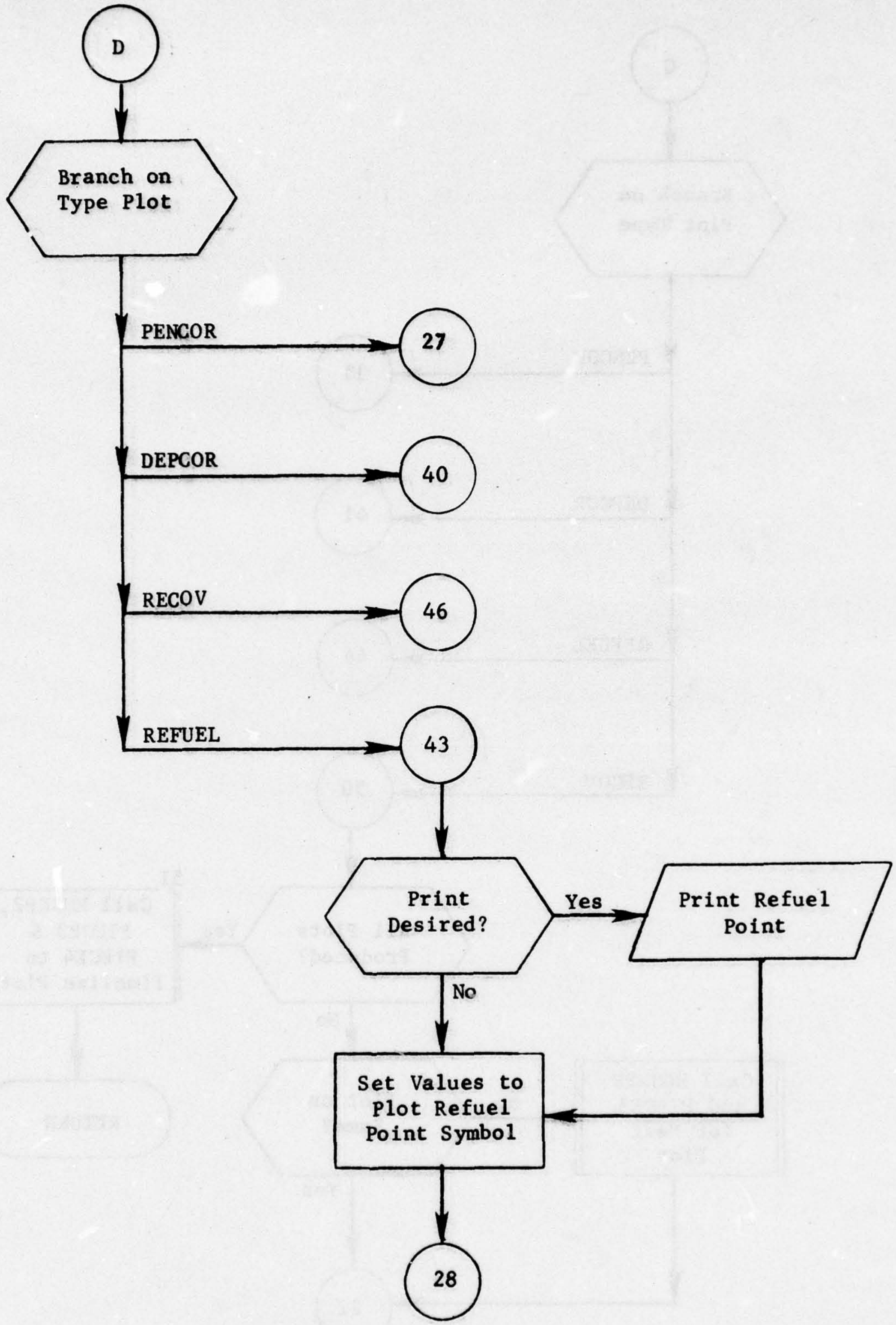


Figure 126. (Part 9 of 17)

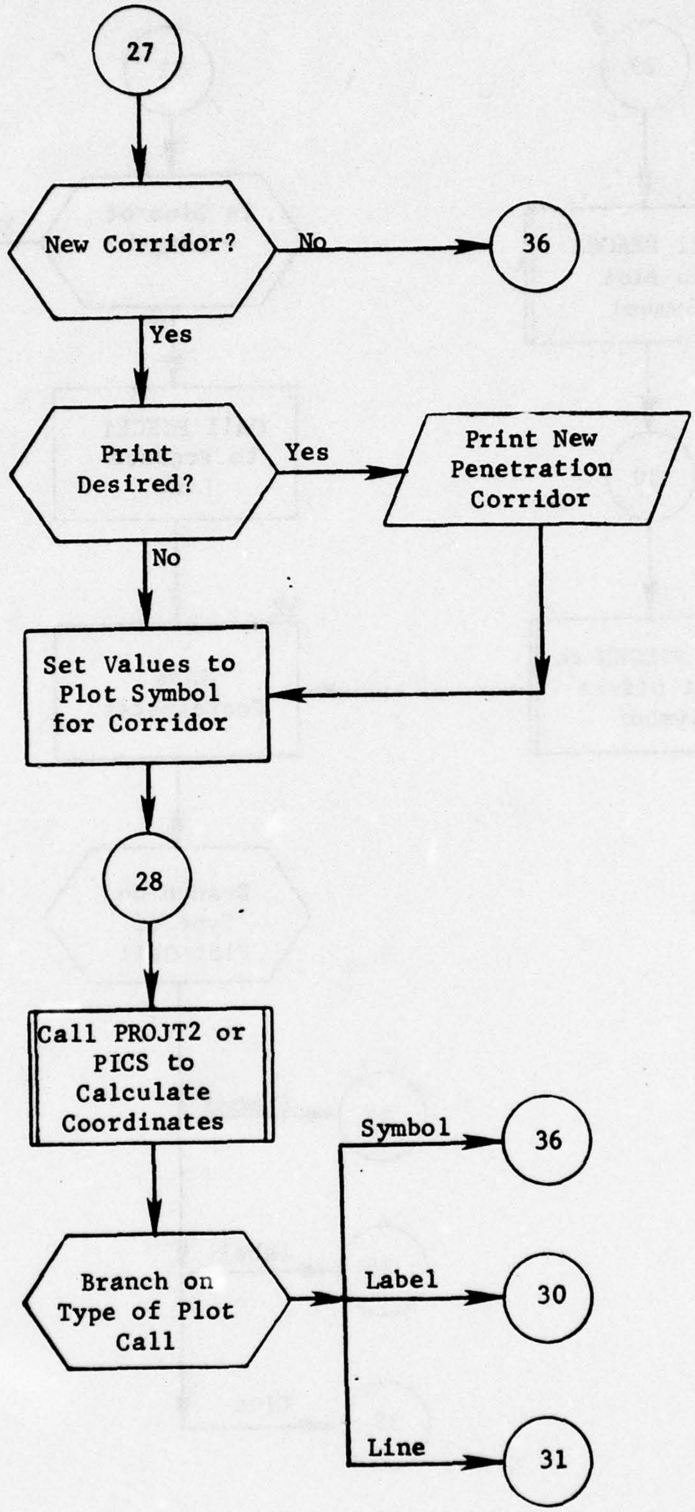


Figure 126. (Part 10 of 17)

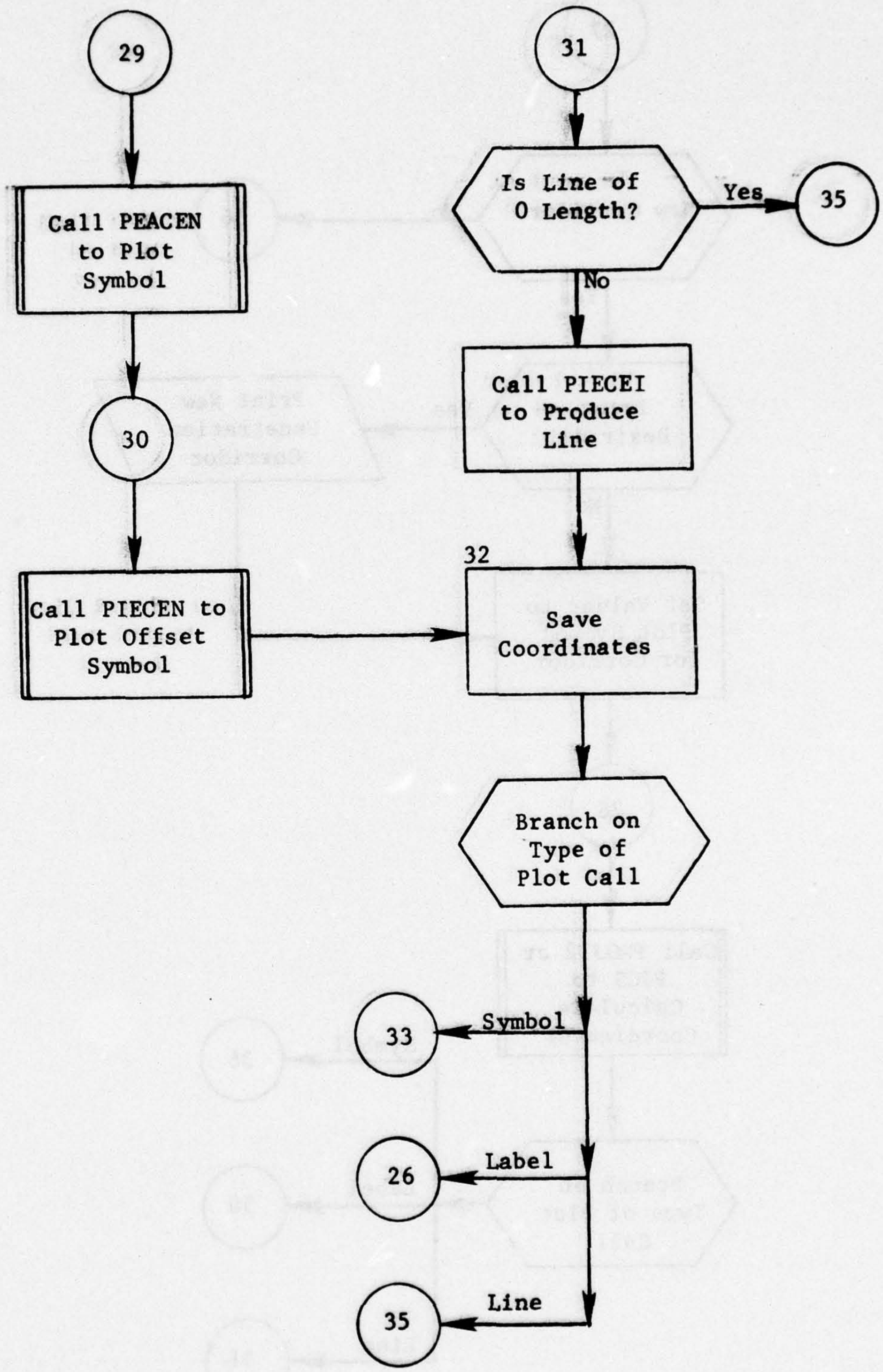


Figure 126. (Part 11 of 17)

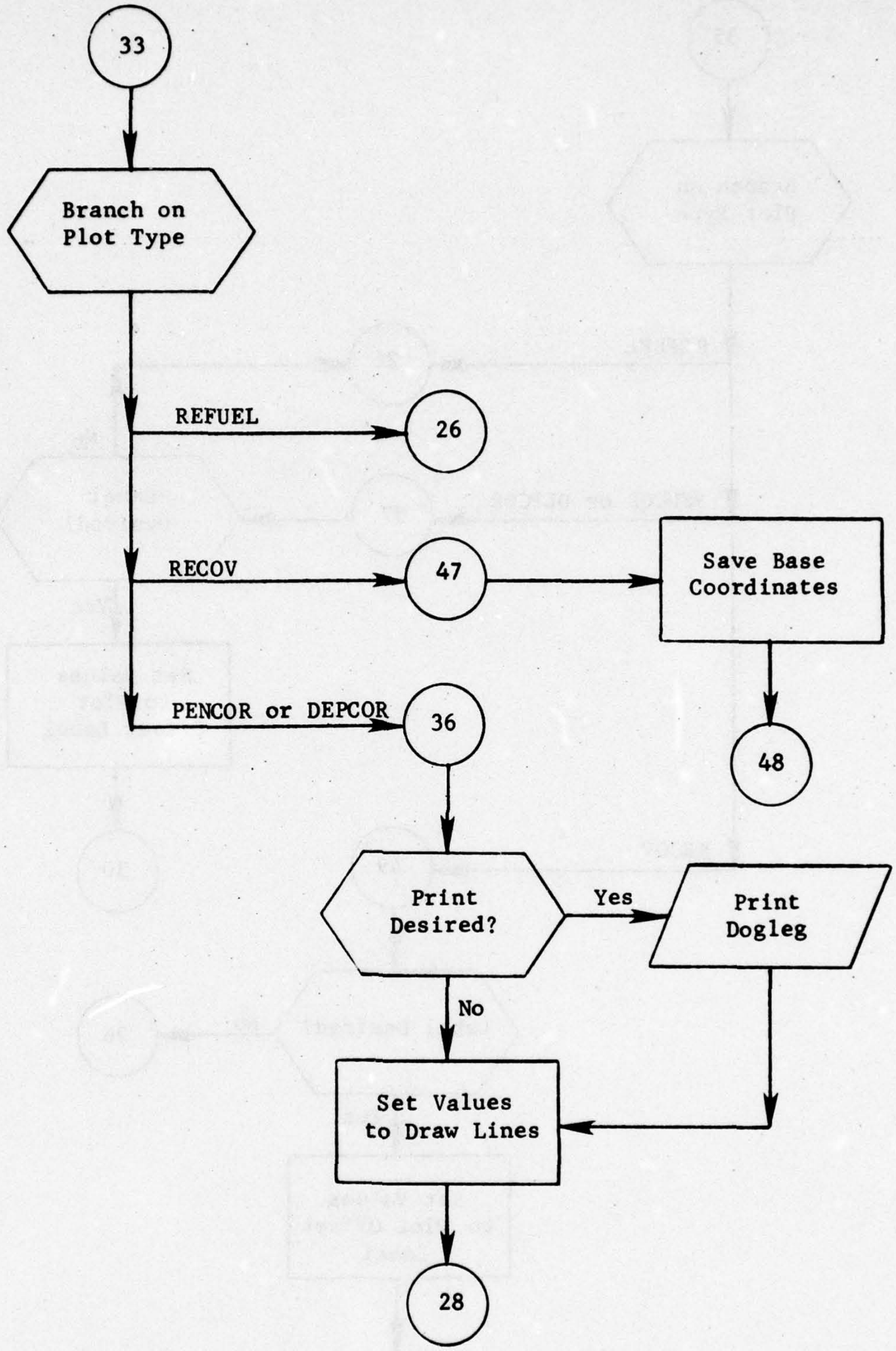


Figure 126. (Part 12 of 17)

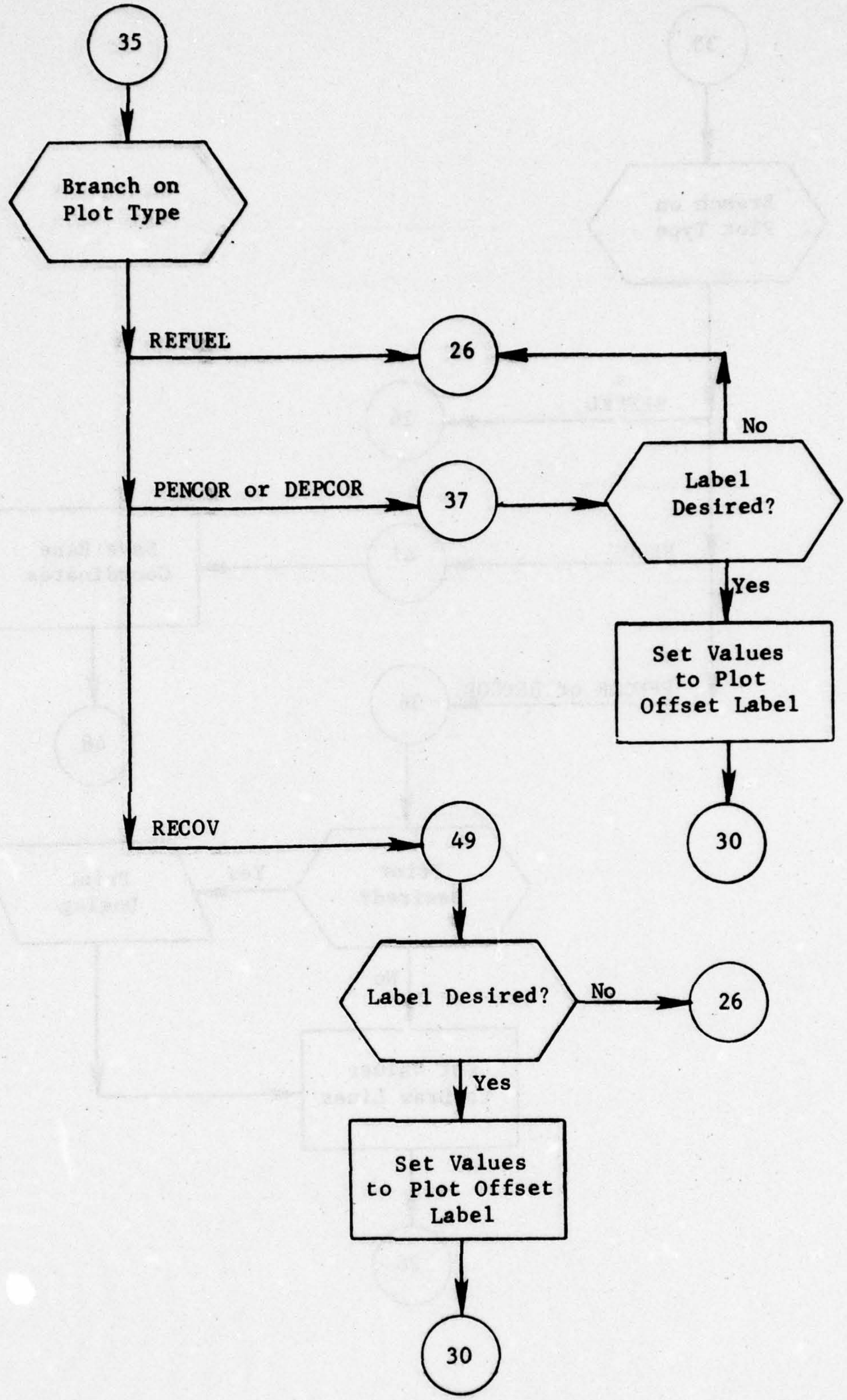


Figure 126. (Part 13 of 17)

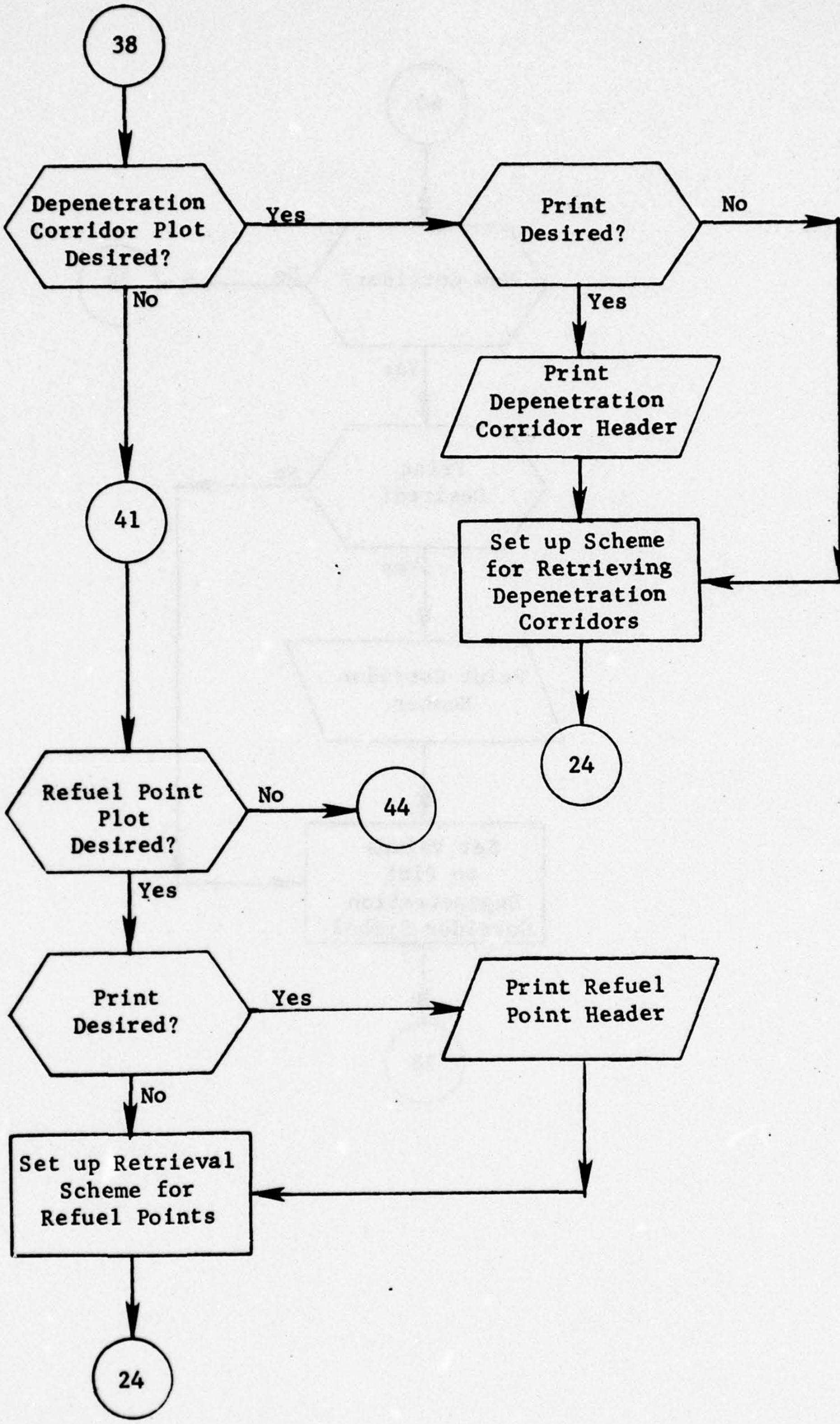


Figure 126. (Part 14 of 17)

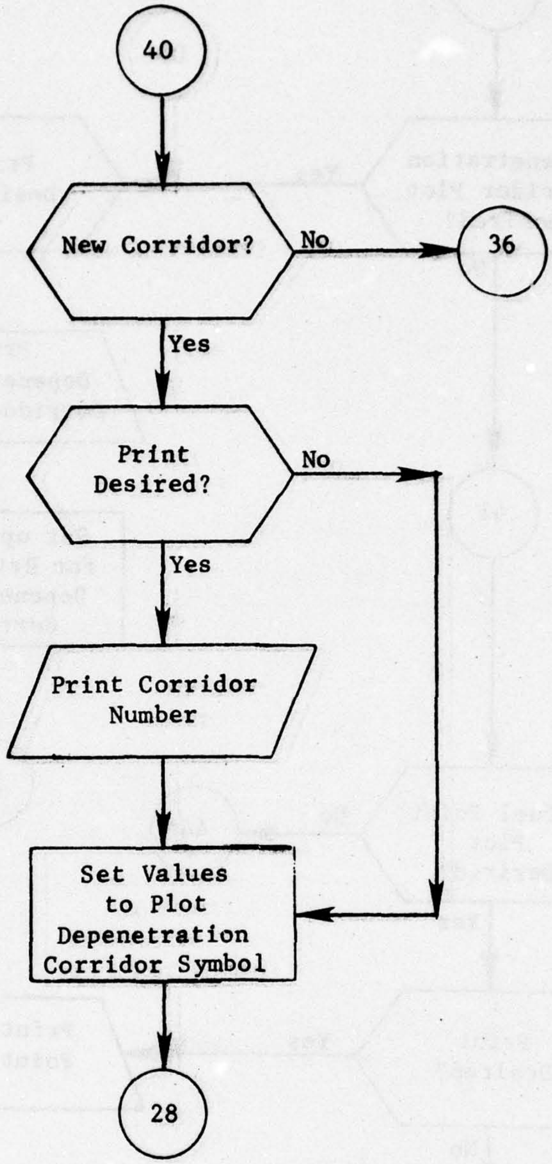


Figure 126. (Part 15 of 17)

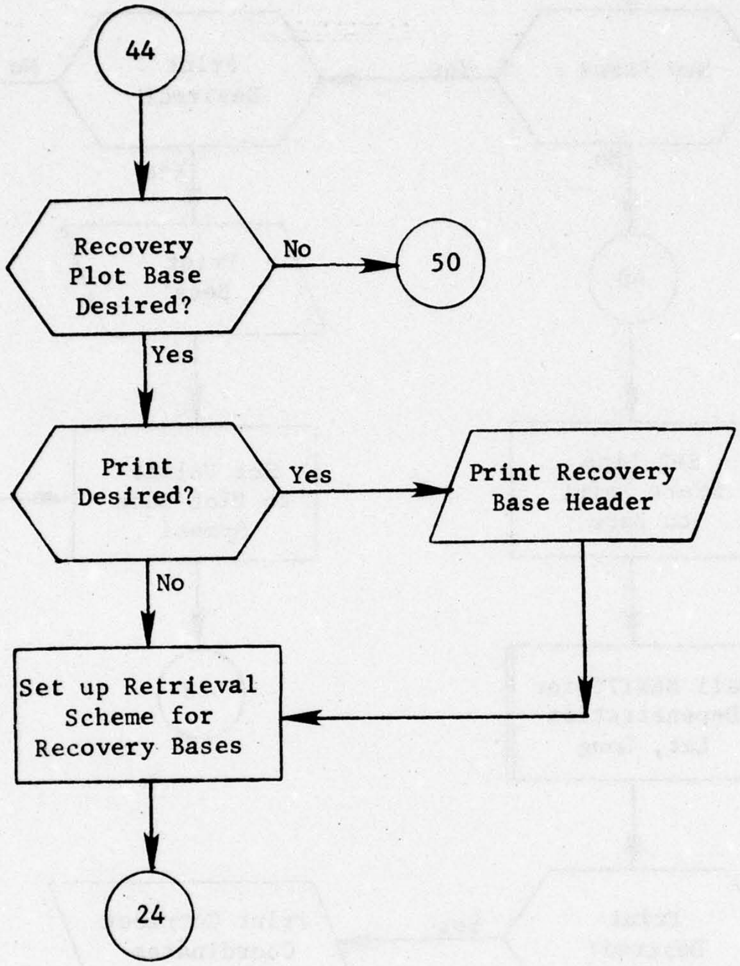


Figure 126. (Part 16 of 17)

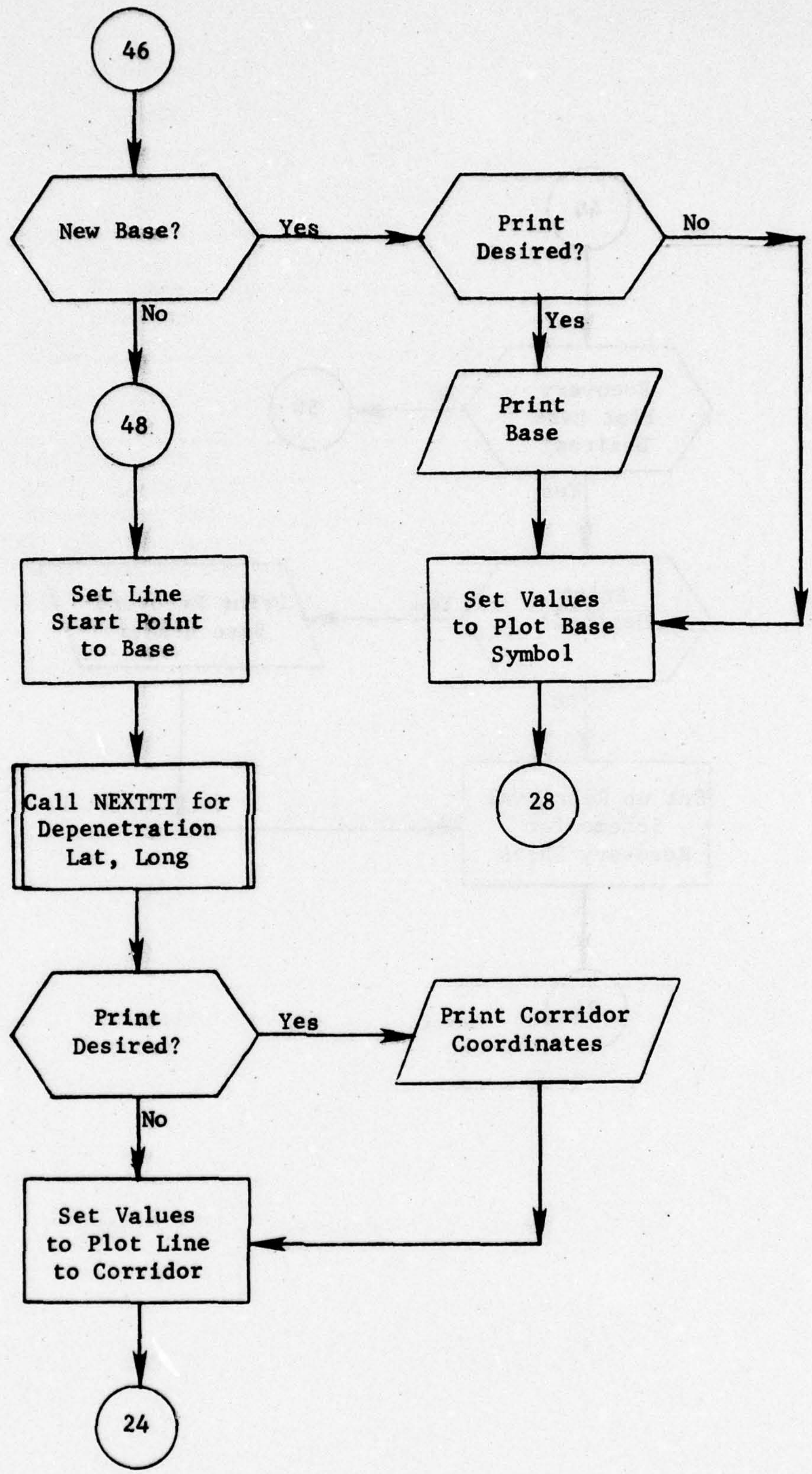


Figure 126. (Part 17 of 17)

8.9.1 Subroutine PICS

PURPOSE: Calculates coordinates for PIC1 and PIC2

ENTRY POINTS: PICS

FORMAL PARAMETERS:

IMAP: Map type - 1 - PIC1
2 - PIC2

HLA: Latitude
HLO: Longitude

COMMON BLOCKS: PRINSP

SUBROUTINES CALLED: None

CALLED BY: PLOTDATA

Method:

The process is to determine which points would fall within the standard plot area and to scale these points properly for the plot. The switch IFLAG is set to one if either latitude or longitude fall outside the plot bounds.

See figure 127 for the precise method.

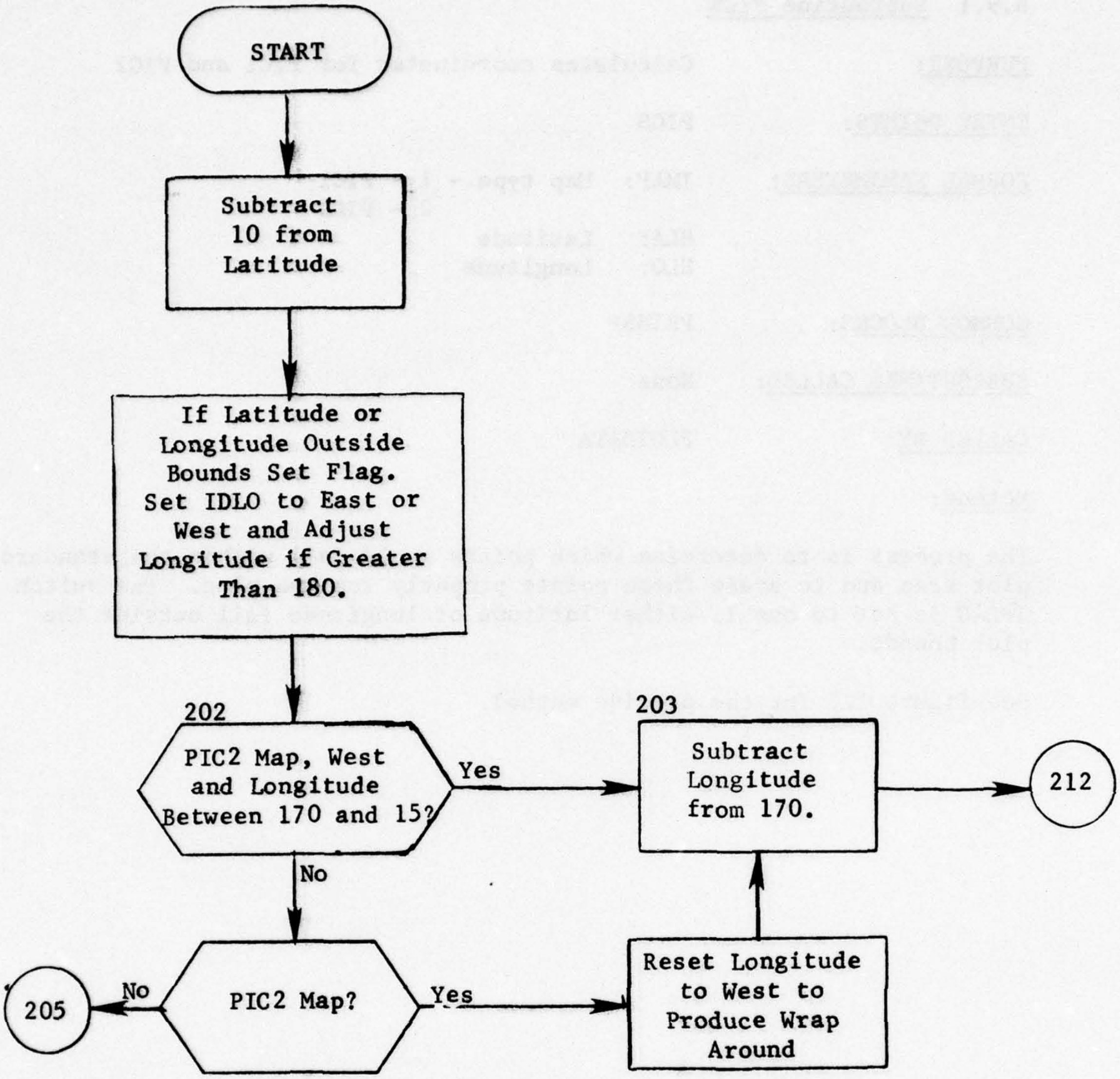


Figure 127. Subroutine PICS (Part 1 of 3)

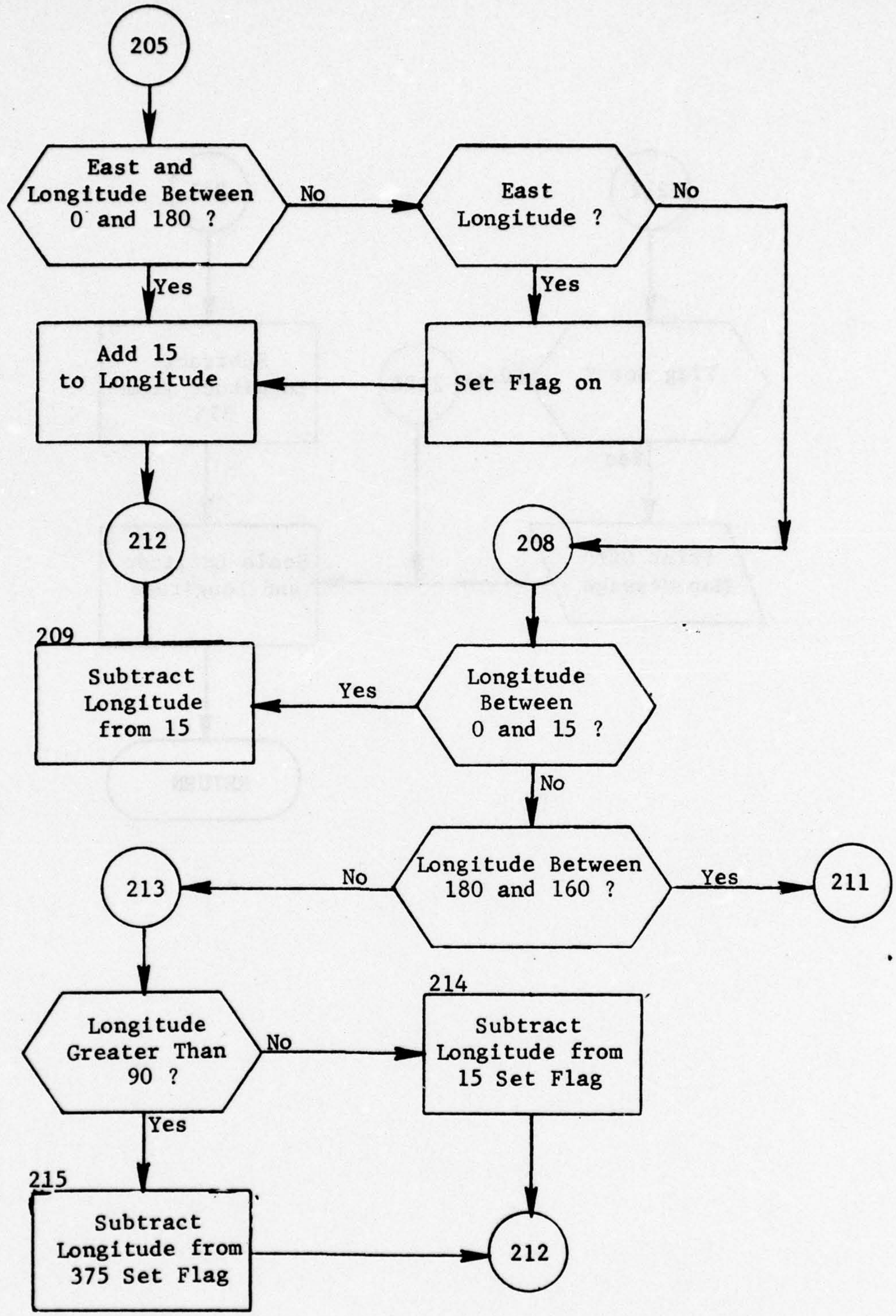


Figure 127. (Part 2 of 3)

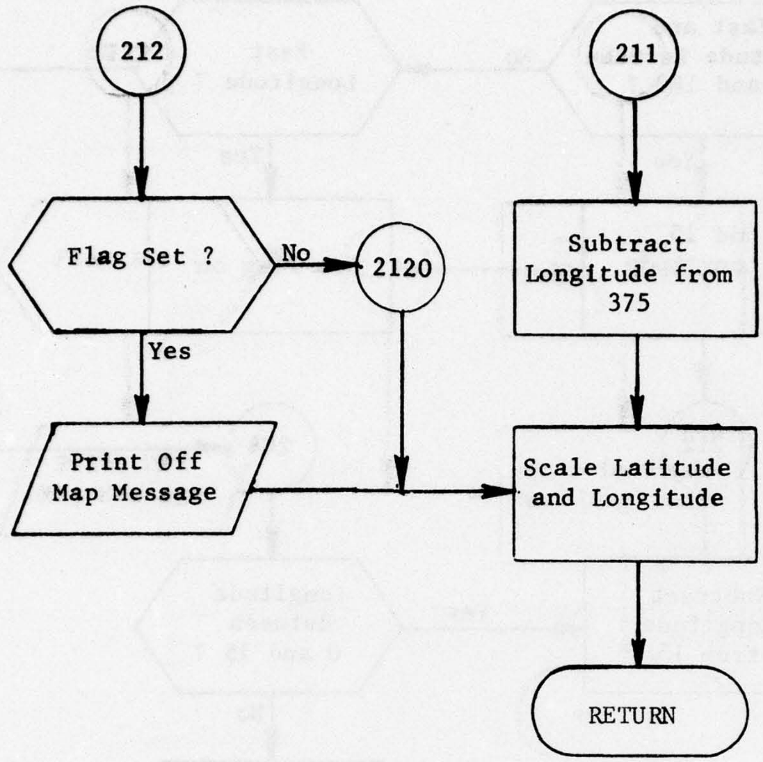


Figure 127. (Part 3 of 3)

8.9.2 Subroutine PROJCT

PURPOSE: Calculate coordinates for PIC3, PIC4 and PIC5

ENTRY POINTS: PROJCT, PROJT2

FORMAL PARAMETERS:
DTORAD: Conversion factor, degrees to radius
SUBHLA: Latitude
SUBHLO: Longitude

COMMON BLOCKS: PLTPRO, PLTSPE, XMEDGE

SUBROUTINES CALLED: None

CALLED BY: PLOTDATA

Method:

Entry PROJCT

First the SCALE is set if it is zero. Next the scaled origin latitude (PORGS) and longitude (BANGL) are calculated. Finally the scaling factor SCLENOW is calculated.

Entry PROJT2

The scaled latitude and longitude are calculated and adjusted according to the origin.

Subroutine PROJCT is illustrated in figure 128.

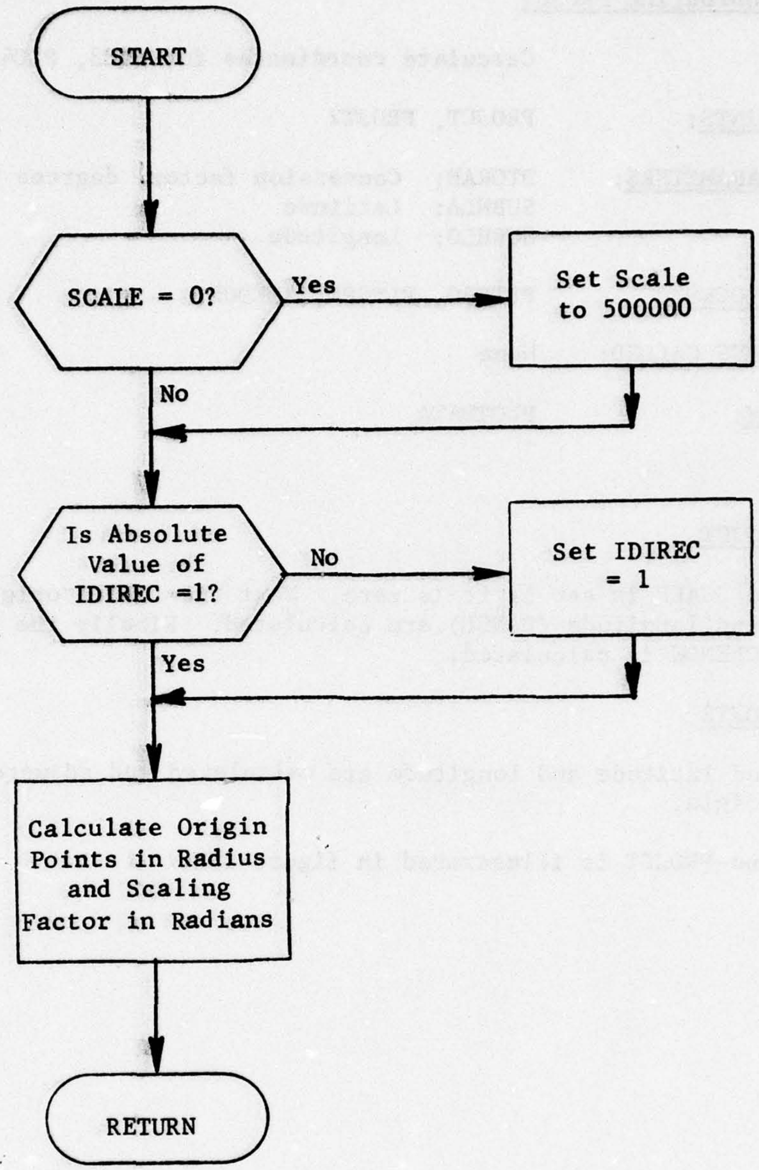


Figure 128. Subroutine PROJCT: Entry PROJCT (Part 1 of 2)

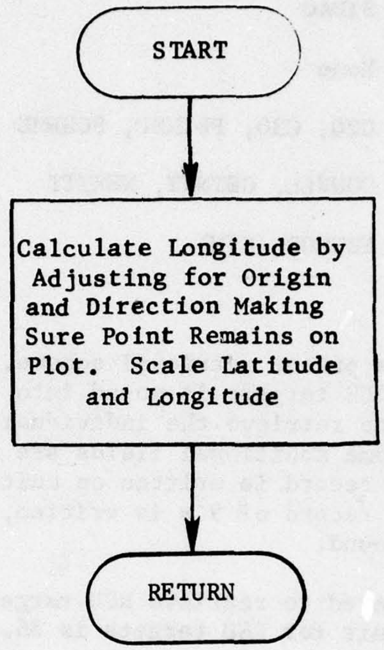


Figure 128. Subroutine PROJCT: Entry PROJT2 (Part 2 of 2)

8.10 Subroutine SIDAC*

PURPOSE: Creates SIDAC output

ENTRY POINTS: SIDAC

FORMAL PARAMETERS: None

COMMON BLOCKS: C20, C30, PRINSP, SCHEME

SUBROUTINES CALLED: CONVLL, GETNXT, NEXTTT

CALLED BY: ENTMOD (EIM)

Method:

Subroutine SIDAC uses a preset retrieval scheme. This scheme which is designed to retrieve BLUE targets is moved into common block SCHEME. Then GETNXT is called to retrieve the individual targets. As each target is retrieved, some additional fields are calculated and then the properly formatted record is written on unit 35. When all targets have been processed, a record of 9's is written, an end of file is added and the tape rewound.

Then the scheme is altered to retrieve RED targets and the process repeats. The output unit for RED targets is 36.

Subroutine SIDAC is illustrated in figure 129.

---

*Main routine of overlay BSIDAC

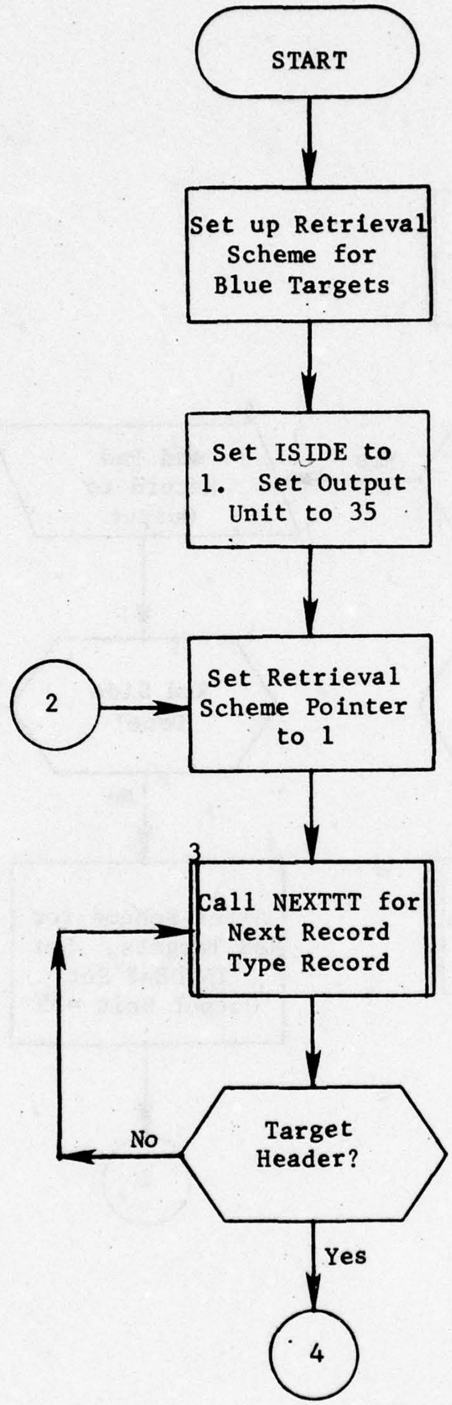


Figure 129. Subroutine SIDAC (Part 1 of 3)

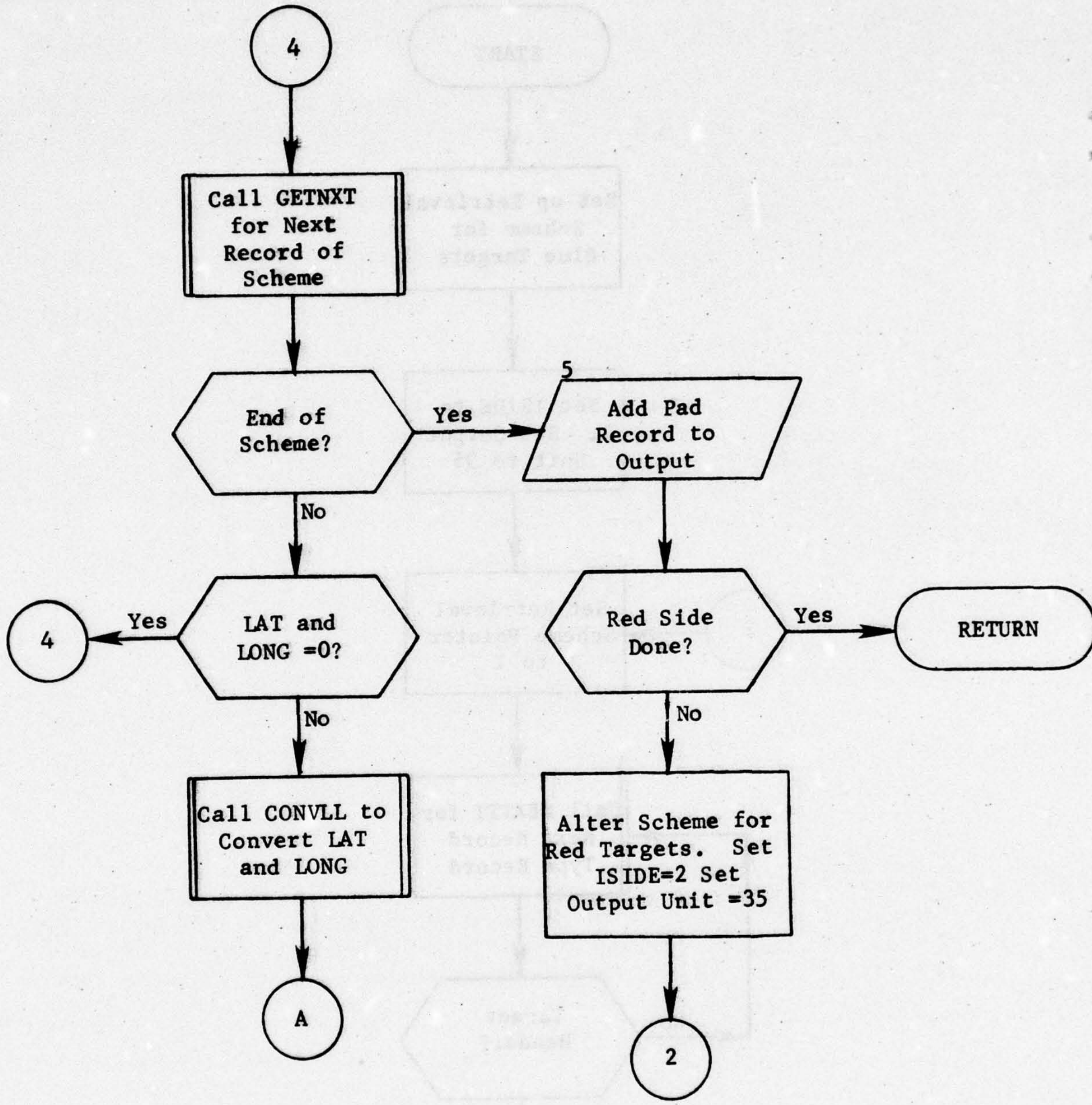


Figure 129. (Part 2 of 3)

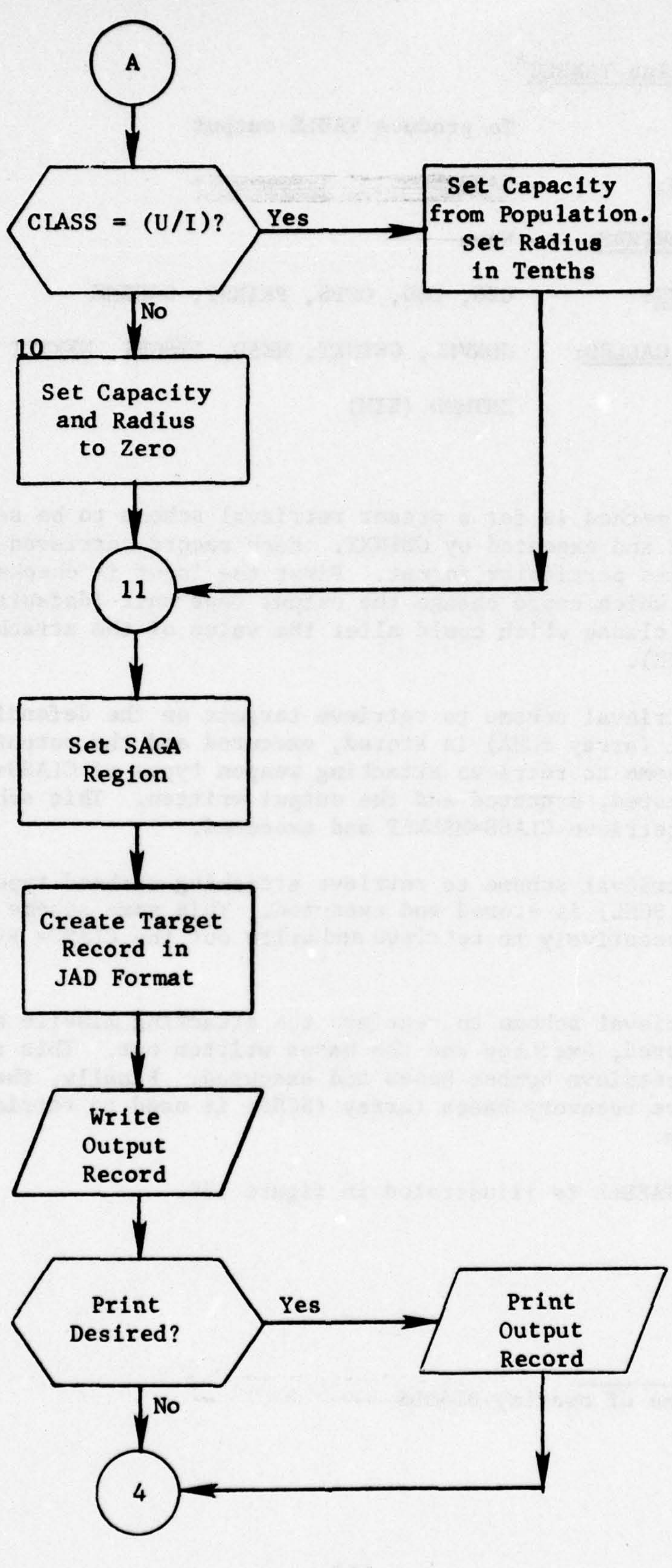


Figure 129. (Part 3 of 3)

8.11 Subroutine TABBLE*

PURPOSE: To produce TABLE output

ENTRY POINTS: TABBLE

FORMAL PARAMETERS: None

COMMON BLOCKS: C20, C30, OOPS, PRINSP, SCHEME

SUBROUTINES CALLED: CONVLL, GETNXT, HEAD, INSGET, NEXTTT

CALLED BY: ENTMOD (EIM)

Method:

The general method is for a preset retrieval scheme to be set in common block SCHEME and executed by GETNXT. Each record retrieved is then written in its particular format. First the input is checked for a UNIT clause which could change the output tape unit (default=35) and for a WHERE clause which could alter the value of the attacking side (default=BLUE).

Next the retrieval scheme to retrieve targets on the defending side of CLASS=MISSIL (array SCHA) is stored, executed and the output written. Then the scheme to retrieve attacking weapon types of CLASS=BMBWEP (SCHB) is stored, executed and the output written. This scheme is then altered to retrieve CLASS=MSLWEP and executed.

Next the retrieval scheme to retrieve attacking warhead types of CLASS= BOMB (array SCHC) is stored and executed. This same scheme is then altered consecutively to retrieve and write out the CLASSs RV, MRV, MIRV and ASM.

Now the retrieval scheme to retrieve the attacking missile bases (array SCHD) is stored, executed and the bases written out. This scheme is altered to retrieve bomber bases and executed. Finally, the scheme for offensive recovery bases (array (SCHE) is used to retrieve and display them.

Subroutine TABBLE is illustrated in figure 130.

*Main routine of overlay BTABLE

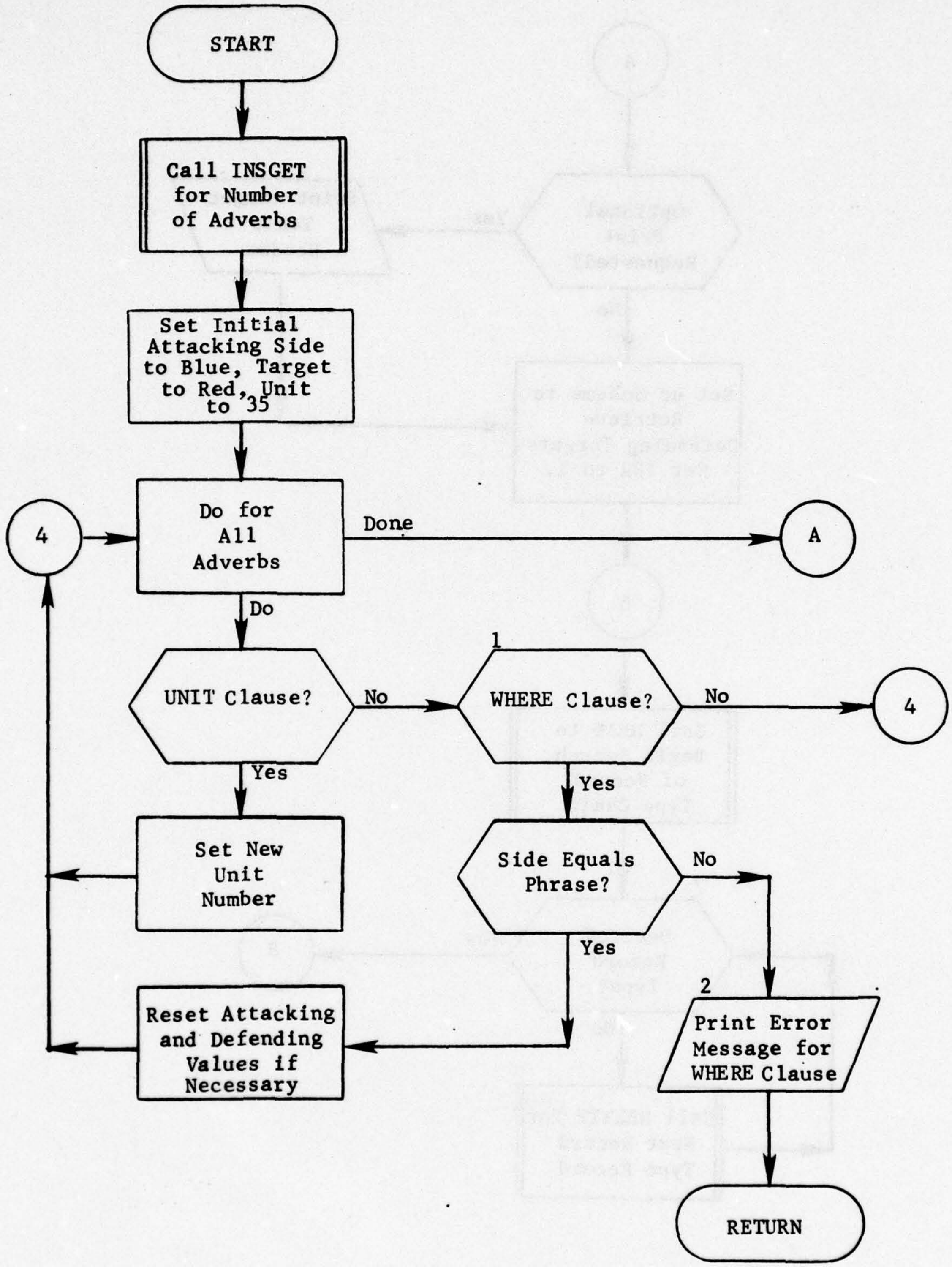


Figure 130. Subroutine TABBLE (Part 1 of 13)

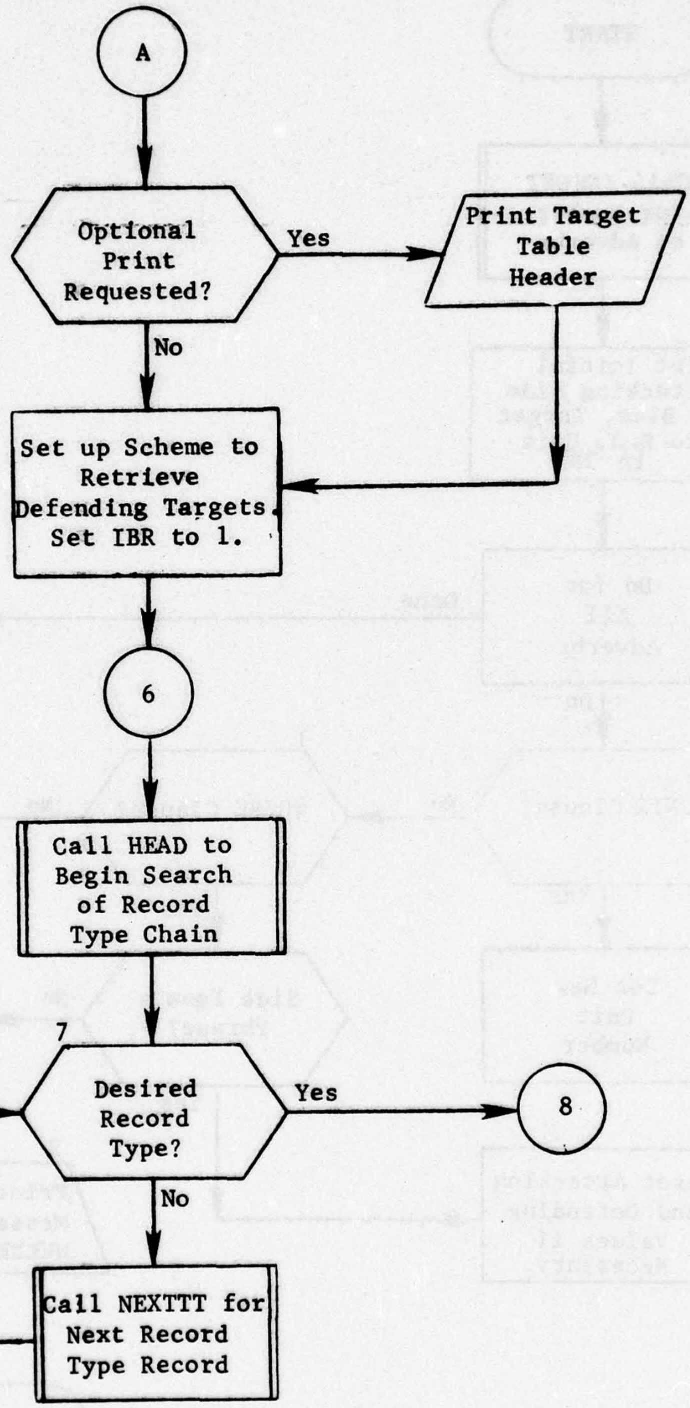


Figure 130. (Part 2 of 13)

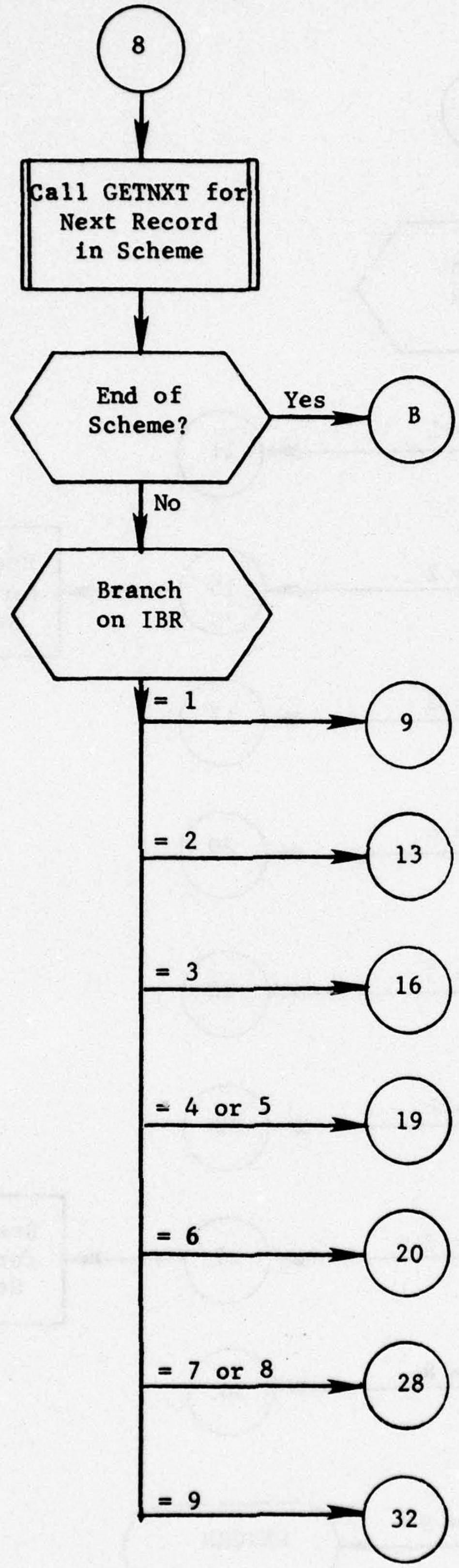


Figure 130. (Part 3 of 13)

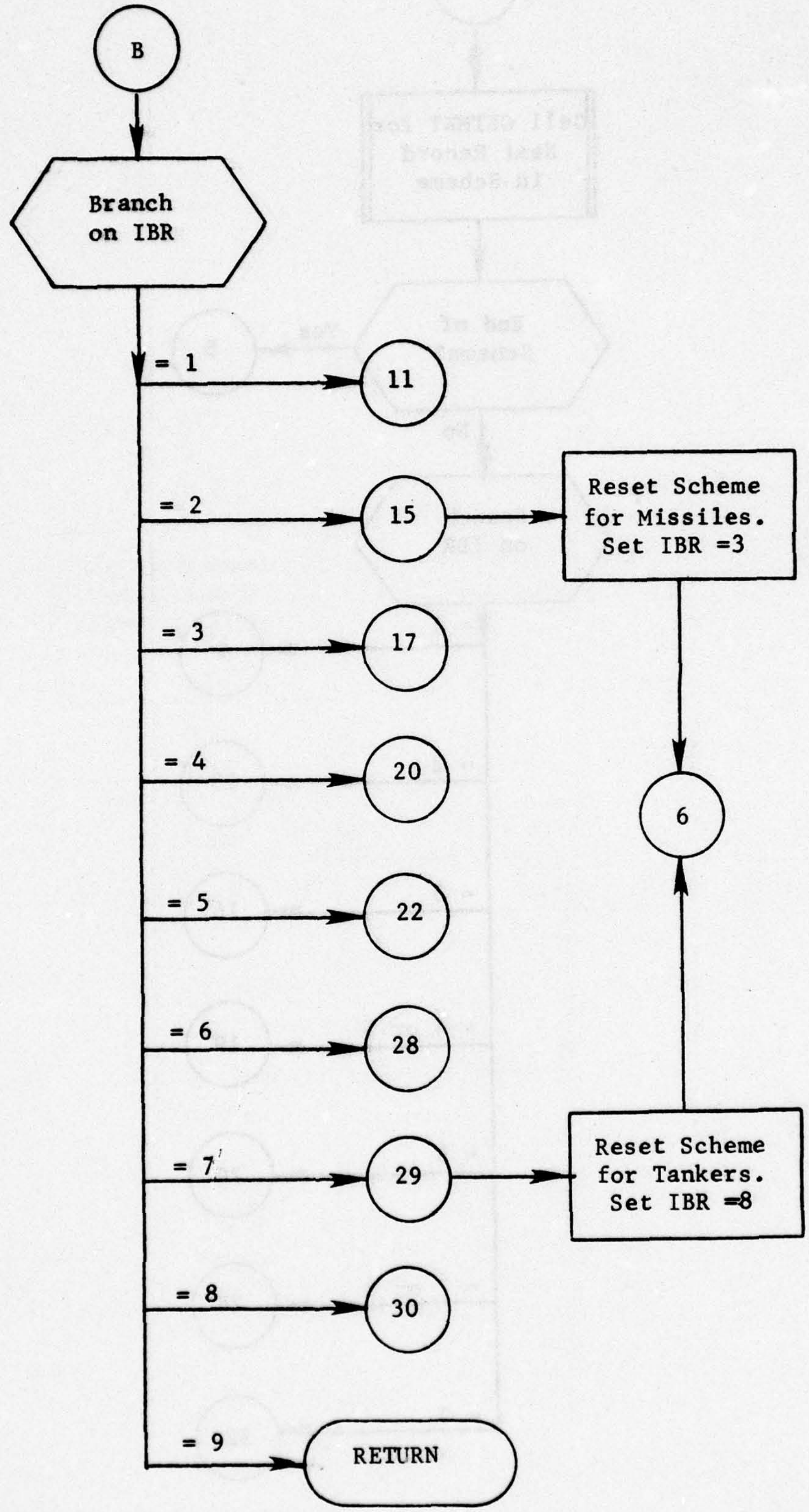


Figure 130. (Part 4 of 13)

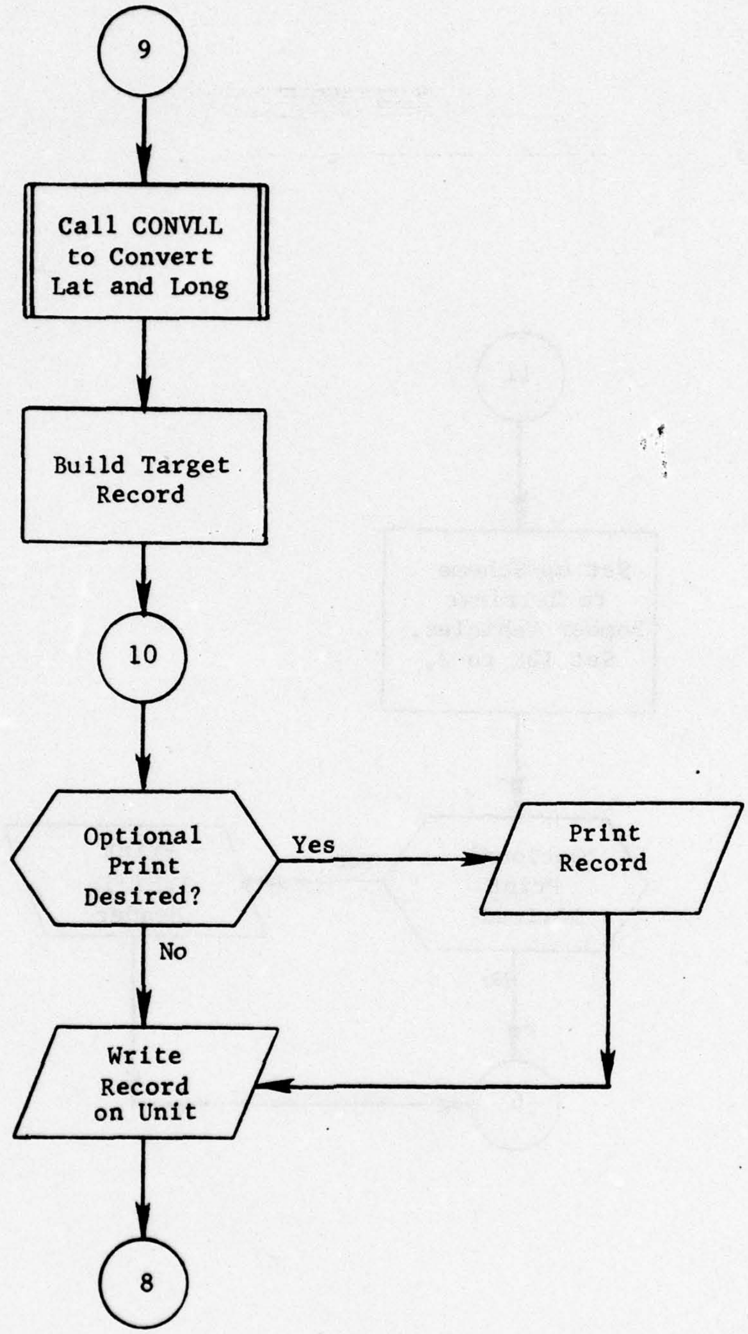


Figure 130. (Part 5 of 13)

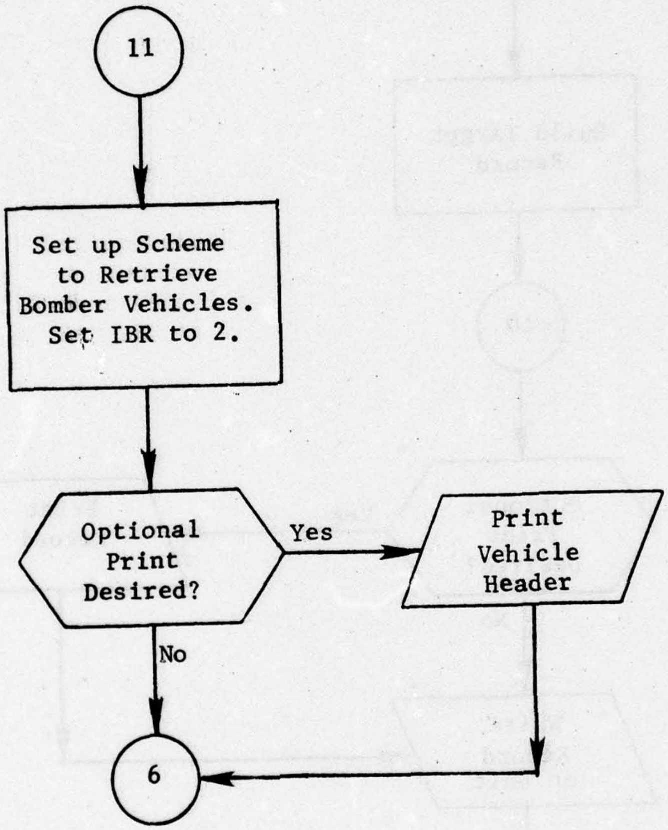


Figure 130. (Part 6 of 13)

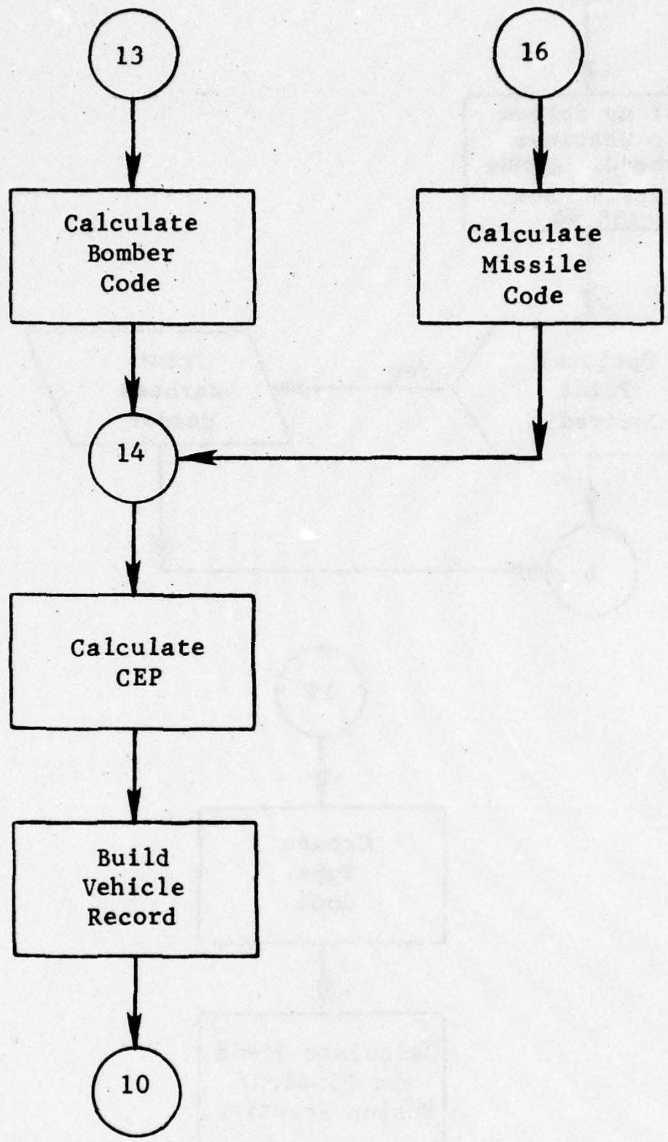


Figure 130. (Part 7 of 13)

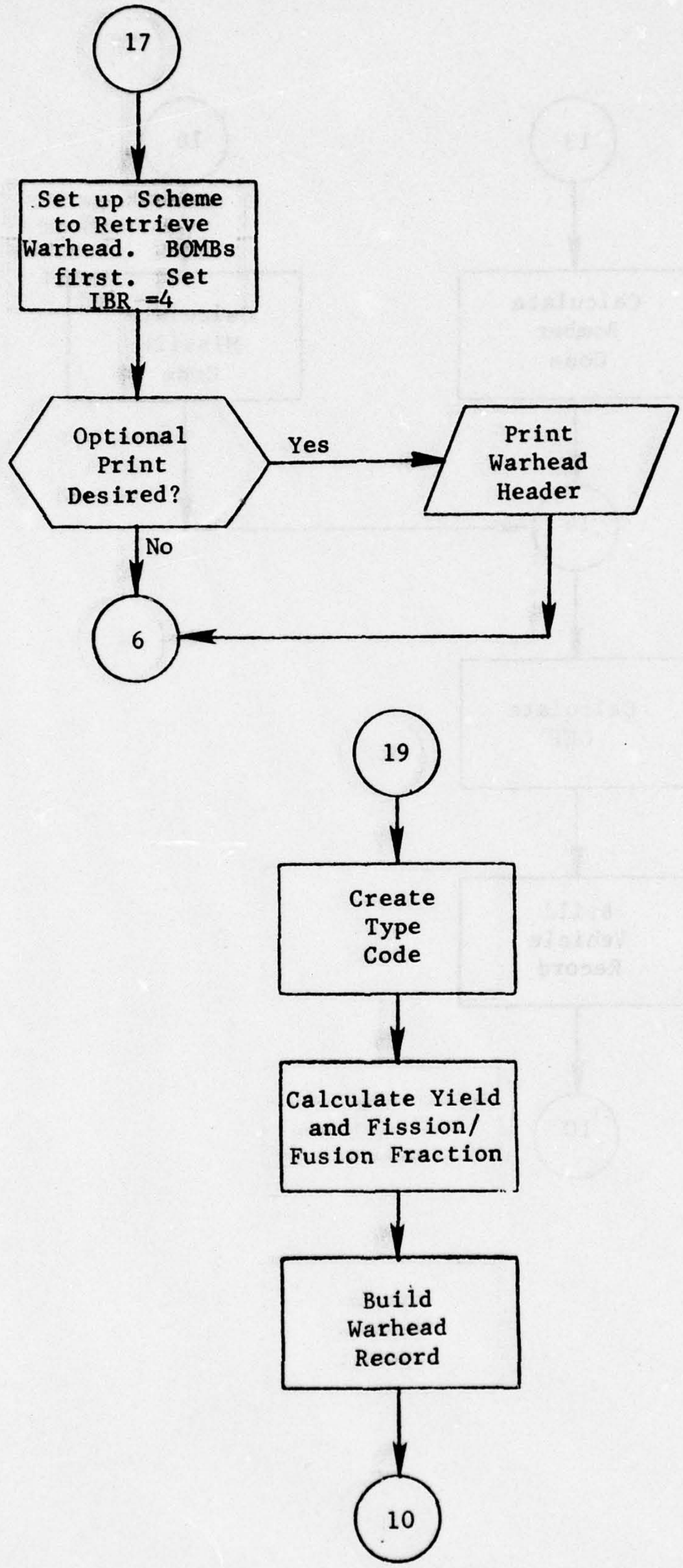


Figure 130. (Part 8 of 13)

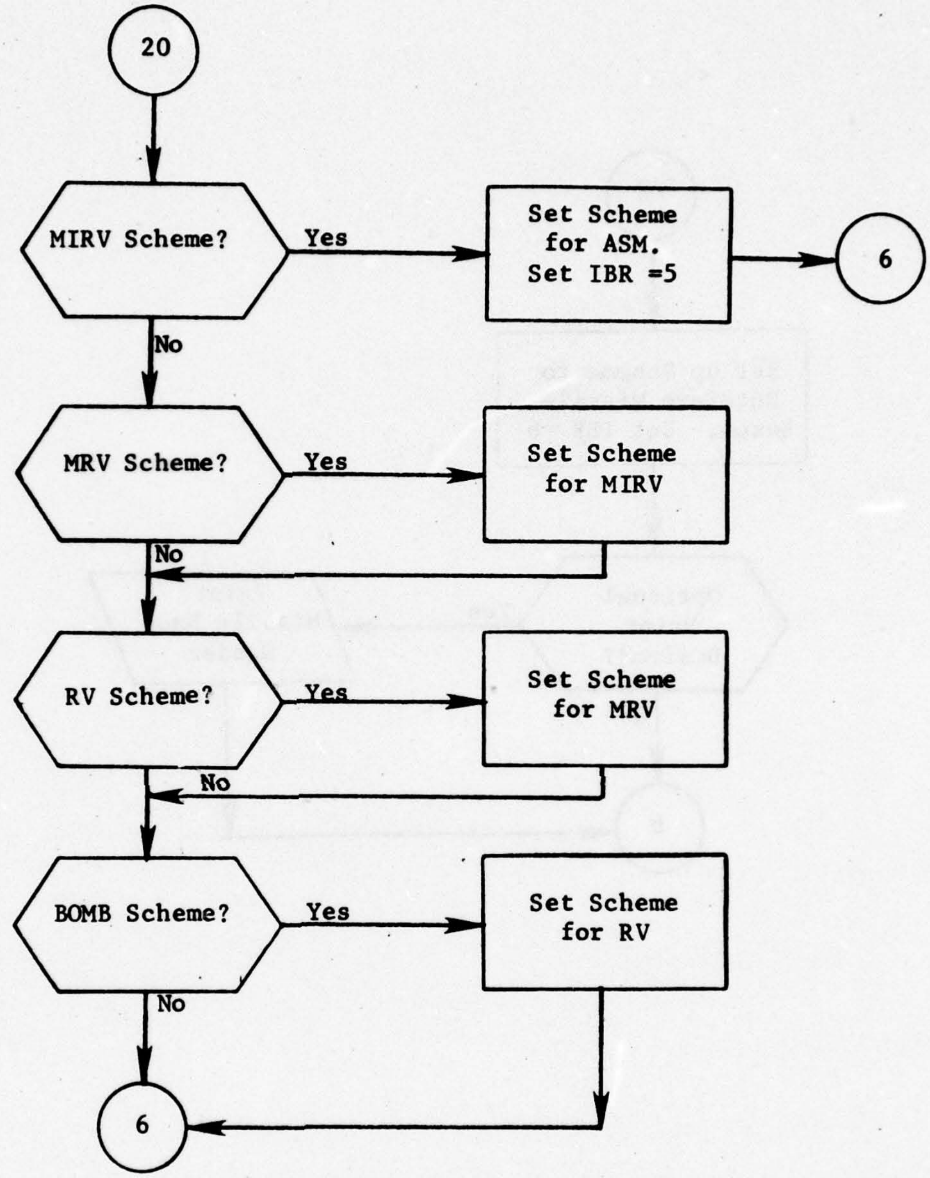


Figure 130. (Part 9 of 13)

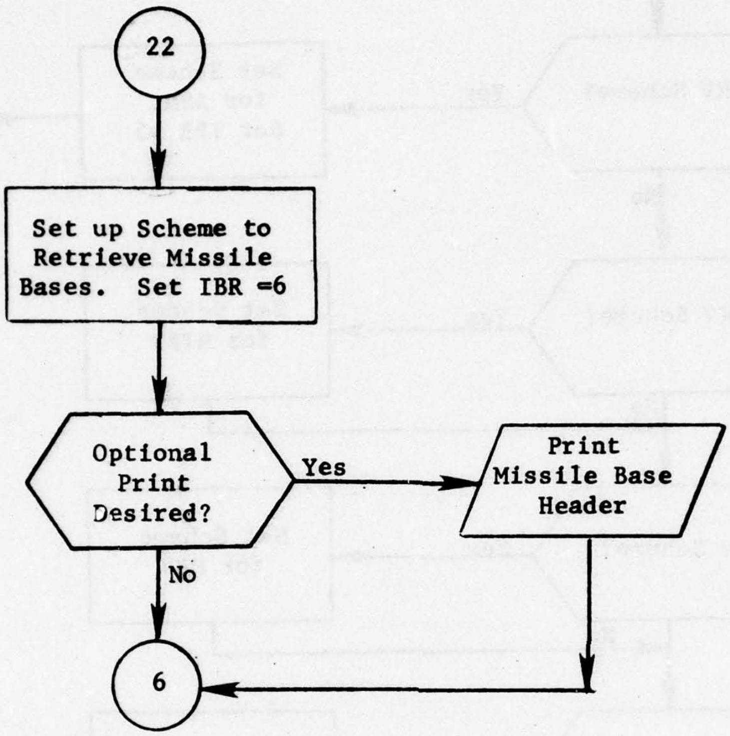


Figure 130. (Part 10 of 13)

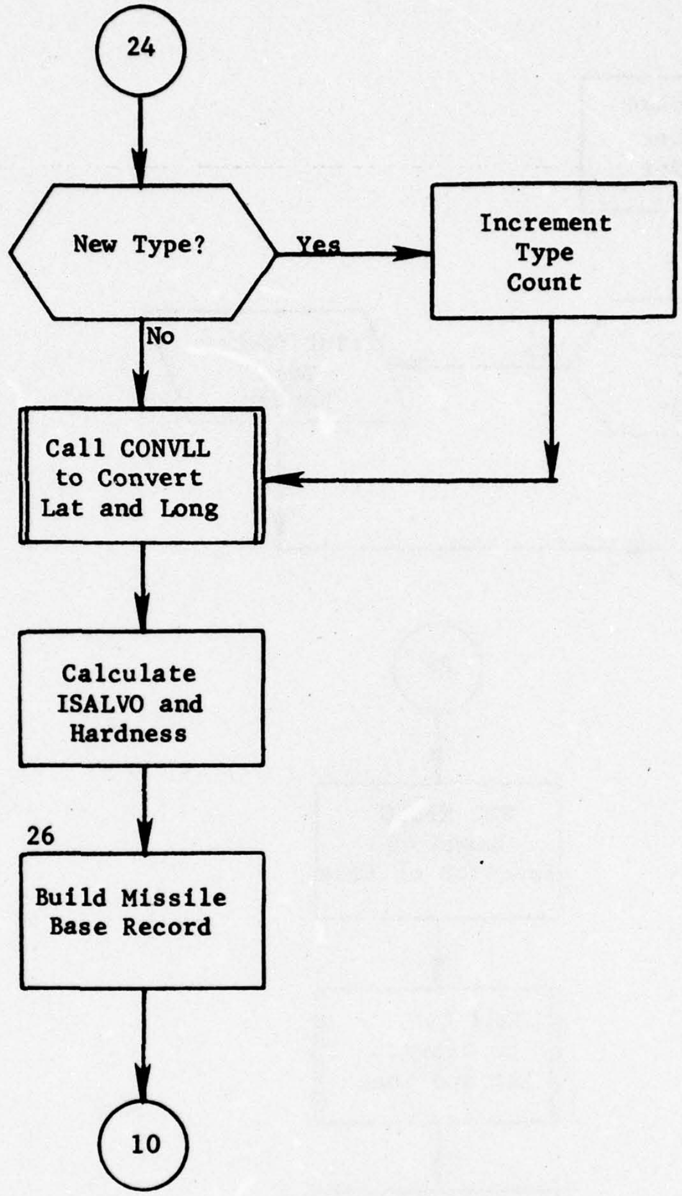


Figure 130. (Part 11 of 13)

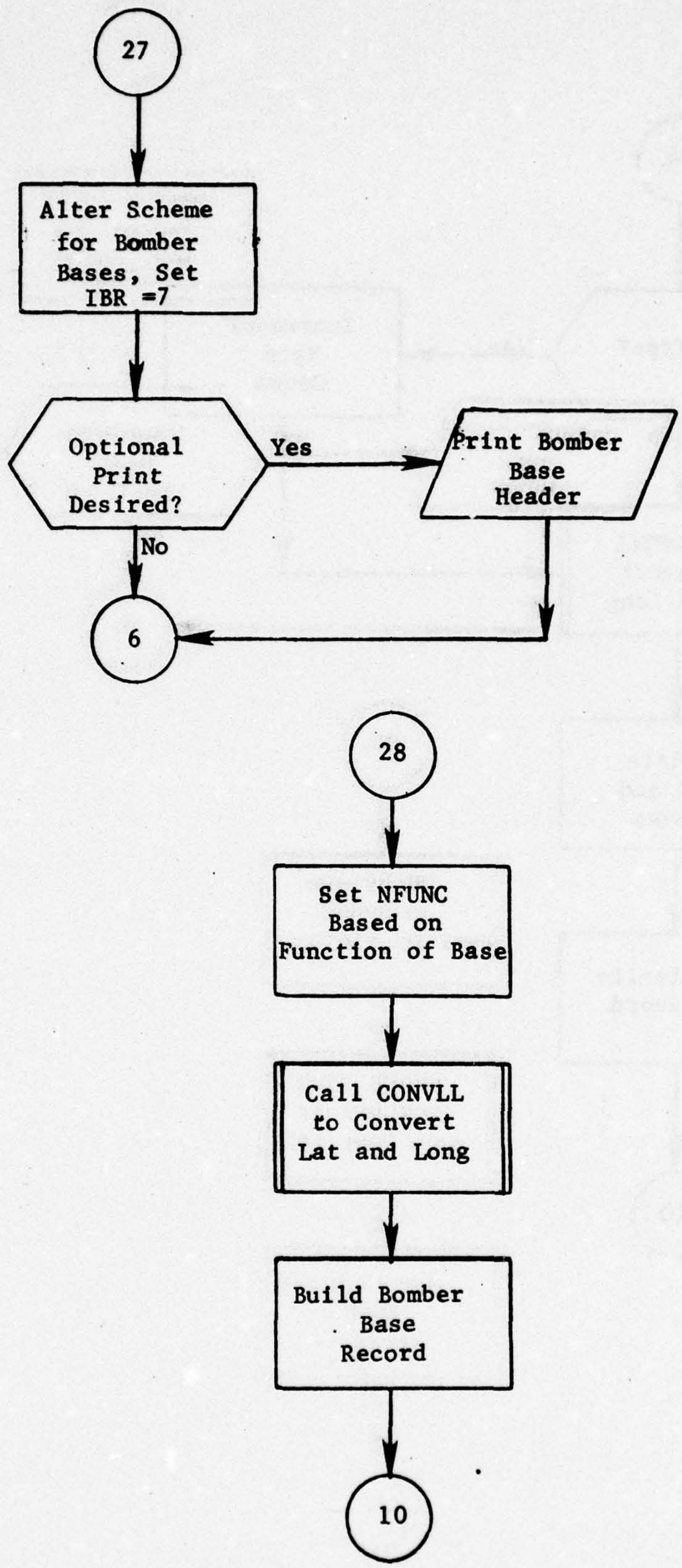


Figure 130. (Part 12 of 13)

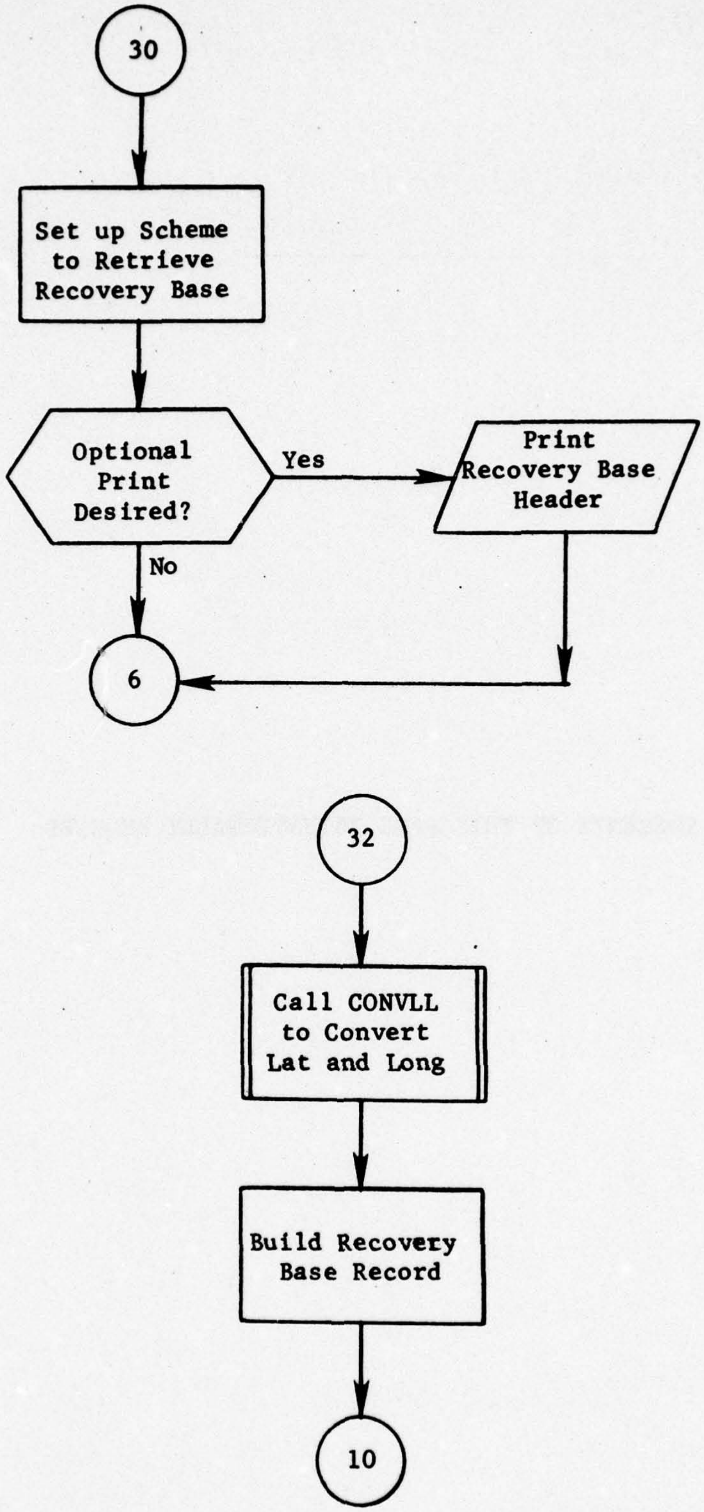


Figure 130. (Part 13 of 13)

Preceding Page Blank - NOT FILMED

SECTION 9. GENERAL UTILITIES

9.1 Purpose

This section contains descriptions of a variety of subroutines and functions, performing various tasks, used throughout the QUICK system. Common blocks used by these programs are described in table 24.

Note that for the majority of subroutines given in this section, the standard subsection referred to as 'CALLED BY' is omitted. This occurs for subroutines that are executed by a large number of calling subroutines and where definition would be quite lengthy and most difficult to maintain.

Table 24. Utility Common Blocks (Part 1 of 5)

Block	Array or Variable	Description
ASNKEY	ASNKEY	Contains reference code of ASNTYP record retrieved by GETTAR.
ATLST		Provides communication with subroutine ATFNDR
	ATNUMB(100)	Attribute's identifying number
	ATADD(100)	Attribute's address
	ATTYP(100)	Attribute's mode (=1, integer, =2, alphabetic, =3, floating point)
	ATRA(100)	Attribute's lower limit
	ATRB(100)	Attribute's upper limit
	NUMAT	Number of attributes
CLASSES	CLASSES(60)	List of values for attribute ACLASS
DATPK	MASK(36)	Data word masks - MASK(i) masks out bits (36-i) through 35)
	ISHTAB(36)	Data word shift - ISHTAB(i) = 2^{i-1}
DEFVAR	VARXX(100)	Value of DEFINE variable
DSPFRM		Provides communication with subroutine FORMAK
	FORMAR(50)	Format being constructed
	IFPNT	Pointer to next character
	IFLNG	Number of words in format
ERRCODE	KABORT	Error code for ABORT (default=61)
	KWARN	Error code for WARNING (default=62)
ERRMESS	IABORT(10)	Error message for ABORT
	IWARN(10)	Error message for WARNING

Table 24. (Part 2 of 5)

Block	Array or Variable	Description
INMAP	INMAP	Indicator if subroutine MAPEDGE has been executed
	XLBL	Plot point number
LMBRT	FL	Lambert projection parameter
	FK	Lambert projection parameter
ORDER	SCHORD(100)	Record type numbers in retrieval scheme order
	SORDNM(100)	Record type name in retrieval scheme order
	LENSCH	Length, in words, of retrieval scheme
PLTPROJ	MERCAT	Projection type
	IDIREC	Indicator of plot direction (+1 for counter - clockwise, -1 for clockwise)
	FLMDAO	Longitude of Origin
	PHIO	Latitude of Origin
	THETAO	Angle between meridian and x-axis
	PHI1	Standard parallel closest to equator
	PHI2	Standard parallel closest to pole
	A1L10	Fractional part of log for PHI1
	A2L10	Fractional part of log for PHI2
PLTSPEC	ISZE	Plot size indicator =0 for 50 x 40, =1 for 20 x 20, =2 for 10 x 10
PLTSPEC	XAXLEN	Length of x-axis in inches
	YAXLEN	Length of y-axis in inches
	FACTOR	Number of plots per page

Table 24. (Part 3 of 5)

Block	Array or Variable	Description
	SCALE	Ratio of world unit to plot units
POLITE	S1	Latitude of beginning interpolation point
	T1	Longitude of beginning interpolation point
	S2	Latitude of interpolation end point
	T2	Longitude of interpolation end point
	FACTOR	Interpolation factor or fraction
	SR	Latitude of interpolated point
	TR	Longitude of interpolated point
PRNSP	PRINON	Switch to control optional prints, = True, produce print, = False, do not produce print
PSCOM		Used to communicate with subroutine PSREC
	RECORD(100)	Body of print/sort record
	RECLEN	Number of words in record
		Used to communicate with subroutine ATFNDR
	RTLIST(100)	List of record type numbers for retrieval scheme
	NUMREC	Number of record types in RTLIST
	HDREC	Record type name of primary header
RTLST	HCLASS	CLASS value of primary header
	HSIDE	SIDE value of primary header
	HOPT	CLASS/SIDE option for retrieval scheme

Table 24. (Part 4 of 5)

Block	Array or Variable	Description
	JHDR	Record type number of primary header
SCHEME	POINT	Pointer to current retrieval scheme instruction
	SCHEME(200)	Retrieval scheme (section 4.4)
SCRTCH	LIST(300)	Storage space used as work area by several subroutines
SIDES	SIDES(5)	List of values for SIDE
SNDMIN	XMIN	Minimum value of x-coordinator
	YATXMN	Y-coordinate at minimum X-coordinate
	XATYMN	X-coordinate at minimum Y-coordinate
	YMIN	Minimum value of y-coordinates
	ISUMIT	Number of points off the graph
SORSCH	SRTSCH(100)	Sort scheme (see section 6.10)
TAPES	PLOTTAPE	Logical unit number for plot tape
	PIECTAPE	Logical unit number for tape for non-plotted points
WAROUT	IWARFL	Logical unit number for printed output
XMEDGE	YMEDGE	Map edge
	XLL	X-coordinate of last point
	YLL	Y-coordinate of last point
	XL	X-coordinate of point to be plotted
	YL	Y-coordinate of point to be plotted

Table 24. (Part 5 of 5)

Block	Array or Variable	Description
	XWEDGE	Converted value for latitude of origin
	BANGL	Converted value for longitude of origin

9.2 Subroutine ABORT

PURPOSE:	Entry ABORT:	To force a core dump on demand.
	Entry WARNING:	To print a warning message and possible error diagnostics.

ENTRY POINTS: ABORT, WARNING

FORMAL PARAMETERS: None

COMMON BLOCKS: ERRCODE, ERRMESS

SUBROUTINES CALLED: FXOPT, FXEM

Method:

This subroutine prints error diagnostics and an optional core memory dump. The error diagnostics include:

Error message
Name of routine (ABORT or WARNING)
A trace of the chain of subprogram calls back to the main program.

The error message is contained in common /ERRMESS/. This block consists of two arrays, each of 10-element length, IABORT and IWARN.

The message to be printed is contained in array IABORT for entry ABORT, and IWARN for entry WARNING. The calling program may place any message in these arrays.

If entry ABORT is used, the error diagnostics will terminate with a Q6 abort and core memory dump, if requested.

Common /ERRCODE/ consists of two error codes, KABORT and KWARN. They are preset to 61 and 62, respectively. These codes are set aside by the operating system as user defined. Each call to this subroutine results in a call to FXOPT to set the abort option followed by a call to FXEM to print the message and execute the option selected.

Subroutine ABORT is illustrated in figure 131.

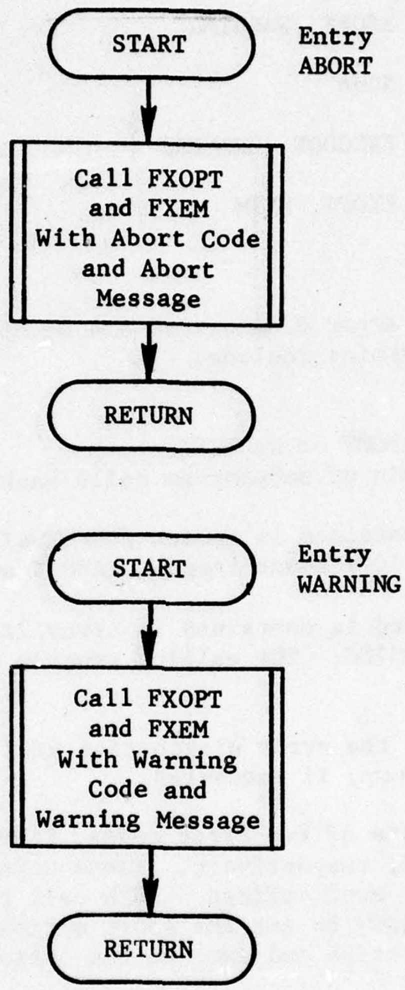


Figure 131. Subroutine ABORT

9.3 Subroutine ATFNDR

PURPOSE: To retrieve attributes information and build a list of record types

ENTRY POINTS: ATFNDR

FORMAL PARAMETERS: None

COMMON BLOCKS: ATLST, C10, C20, C30, OOPS, RTLST

SUBROUTINES CALLED: HDFND, HEAD, NEXTTT, PRIMHD

CALLED BY: BLDOTH, BUILDTAB, DSPMAK

Method:

For a discussion of the method see also section 4.4.

Step One

The ATRIB chain is retrieved. Each member of the chain is matched against the input attribute list ATNUMB. When a match is found the address, type, lower and upper range are stored in common block ATLST (ATADD, ATTYP, ATRA, ATRB, respectively). Next a branch is made on the type of the attribute vis-a-vis its presence on record types (see section 4.4). Single attributes have their record types saved in RTLIST. Control attributes have their controlled record type saved in RTLIST. Multiple attributes are saved in MLAT.

Step Two

The primary header is determined either by using the value of HCLASS (if it is nonblank) in a call to HDFND or by using the highest numbered record type in a call to PRIMHD. The primary header is added to RTLIST.

Step Three

The ATRIB chain is now searched for the multiple attributes (MSAT). For each attribute in MLAT, RTLIST is compared to its list of record types. If a match is found the process takes no action.. If no match is found all the record types for the multiple attribute are saved in MLTLST.

Step Four

Now a 'chain down' process is used starting with the primary header. At each level of the hierarchy, all those records types which are details of chains for which the record type in question (i.e., the

primary header) is master are compared to the list of multiple attribute record types (MLTLST). Any matchs are added to RTLIST. Then the first record type where a match was found is used as the master for the next hierarchical level and so on.

Subroutine ATFNDR is illustrated in figure 132.

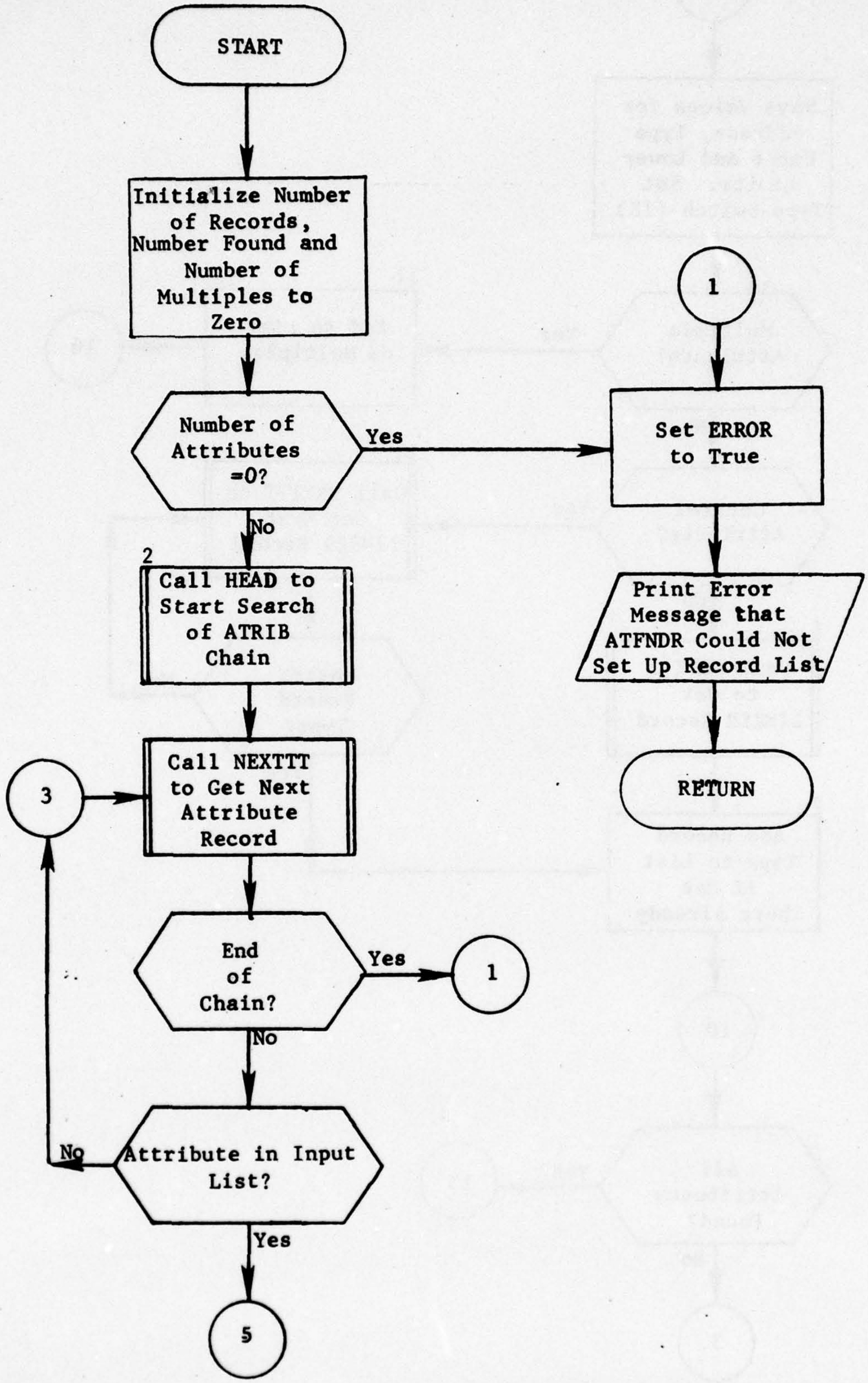


Figure 132. Subroutine ATFNDR (Part 1 of 6)

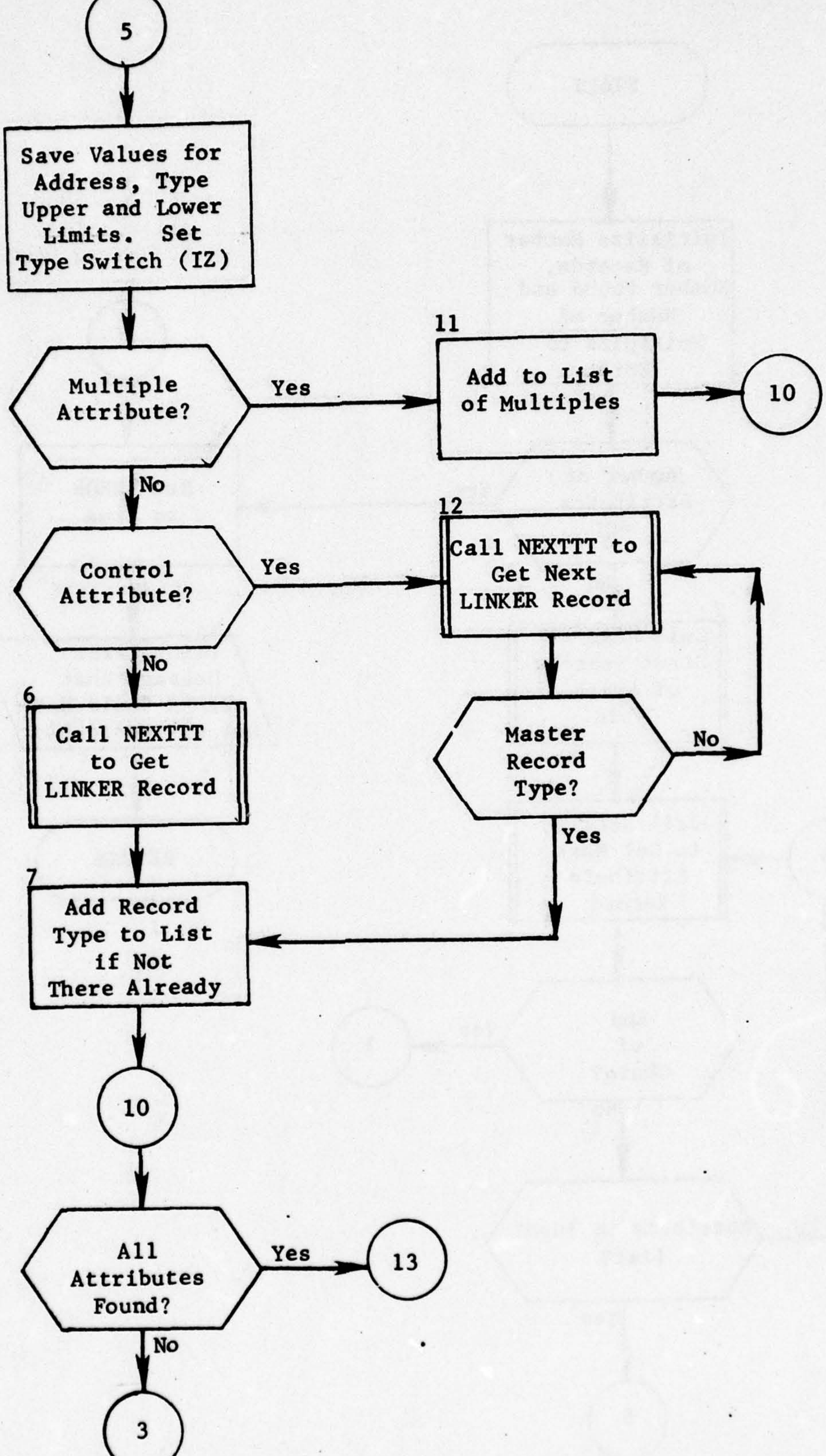


Figure 132. (Part 2 of 6)

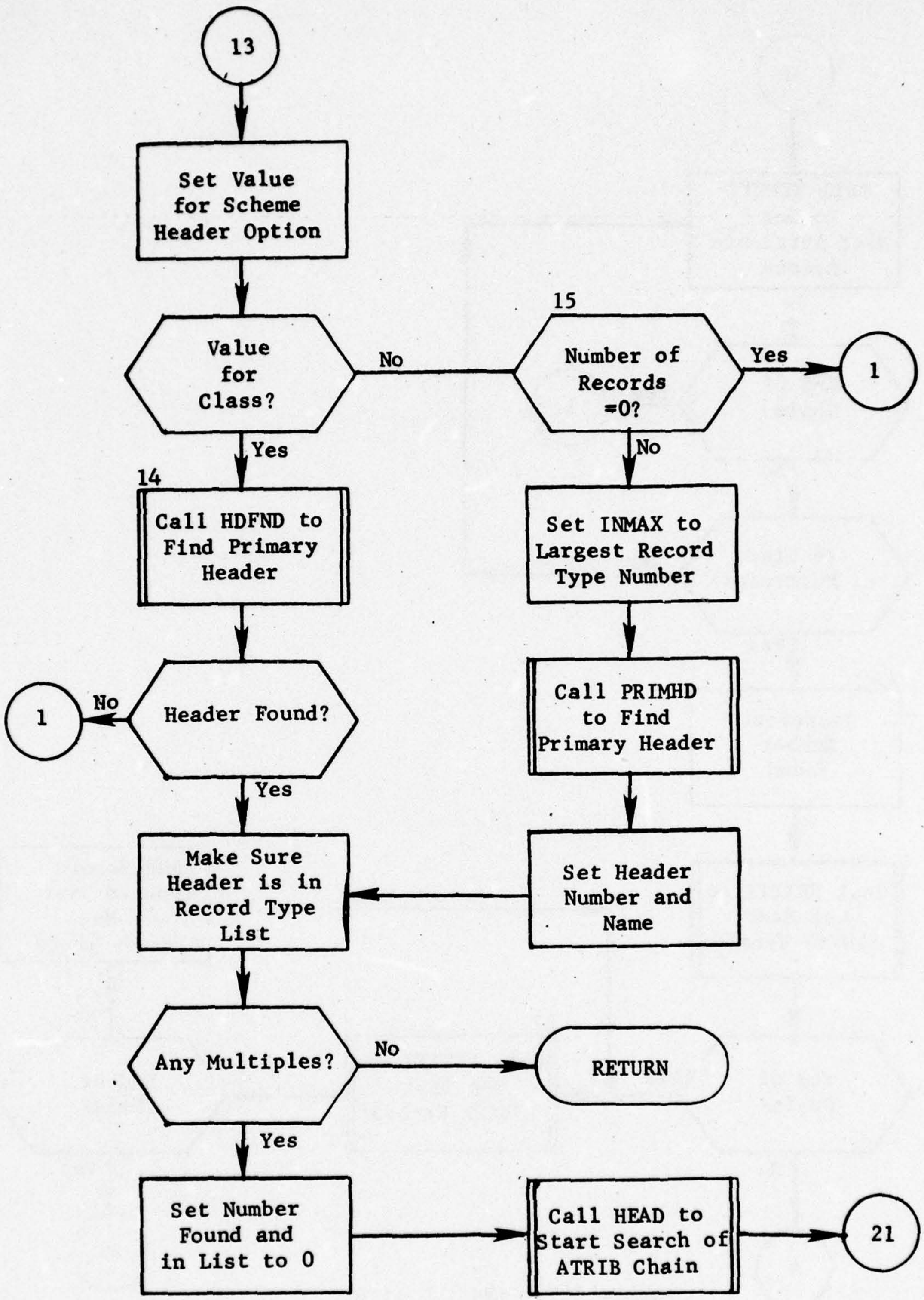


Figure 132. (Part 3 of 6)

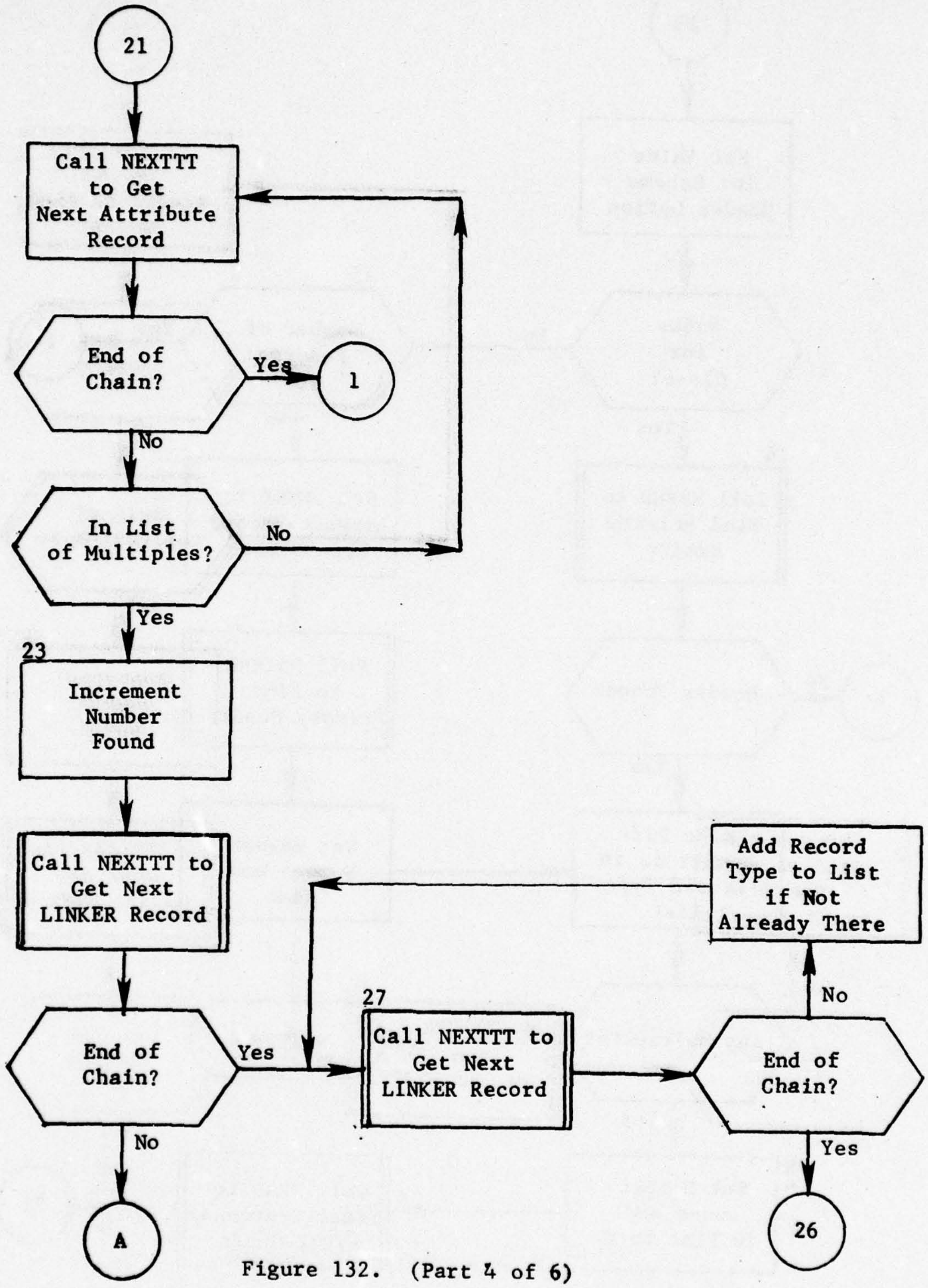


Figure 132. (Part 4 of 6)

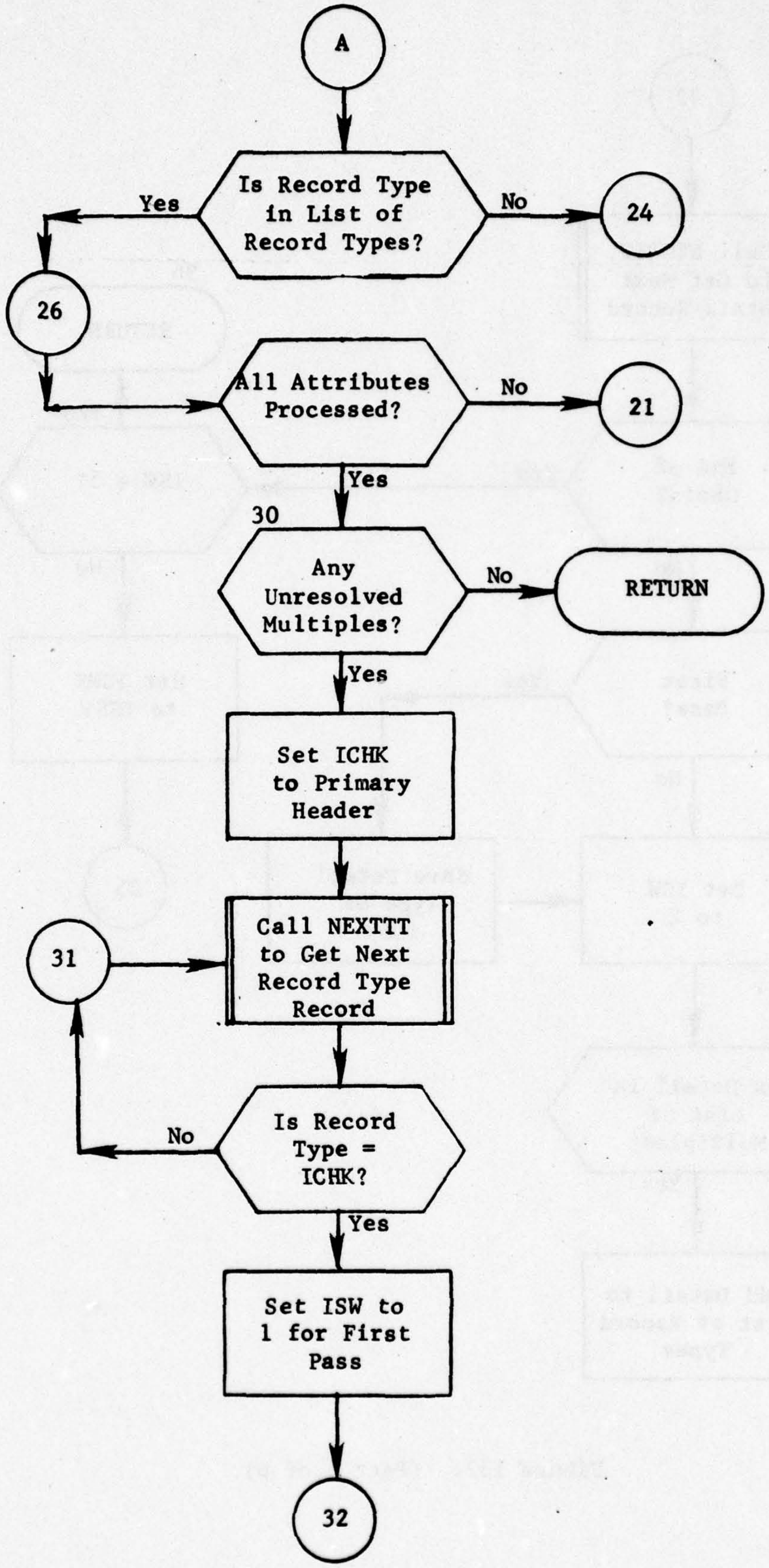


Figure 132. (Part 5 of 6)

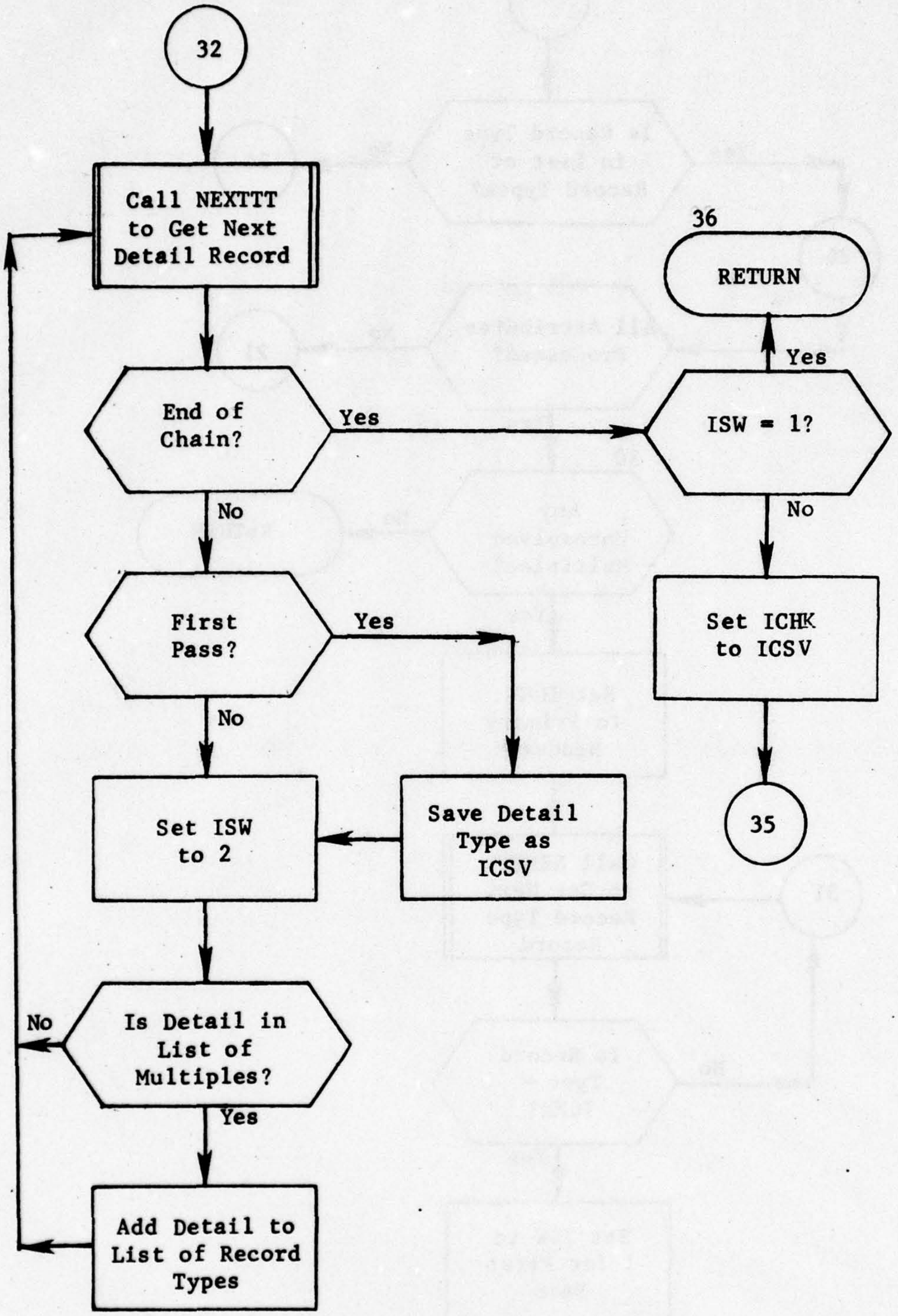


Figure 132. (Part 6 of 6)

9.4 Function ATN2PI

PURPOSE: To calculate the arc tangent function over the interval 0 to 2π.

ENTRY POINTS: ATN2PI

FORMAL PARAMETERS: Y, X (floating point numbers)

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method:

This function calculates the arc tangent of the value (Y/X). The arc tangent returned to the calling program lies within the interval from 0 to 2π. The operating system function ATAN returns the principal value of the arc tangent; i.e., over the interval $-\pi/2$ to $+\pi/2$. ATN2PI uses ATAN to compute the principal value of the arc tangent of Y/X. The signs of X and Y are investigated to determine the quadrant of the arc tangent. The principal value is then modified to return a value within the correct quadrant.

Function ATN2PI is illustrated in figure 133.

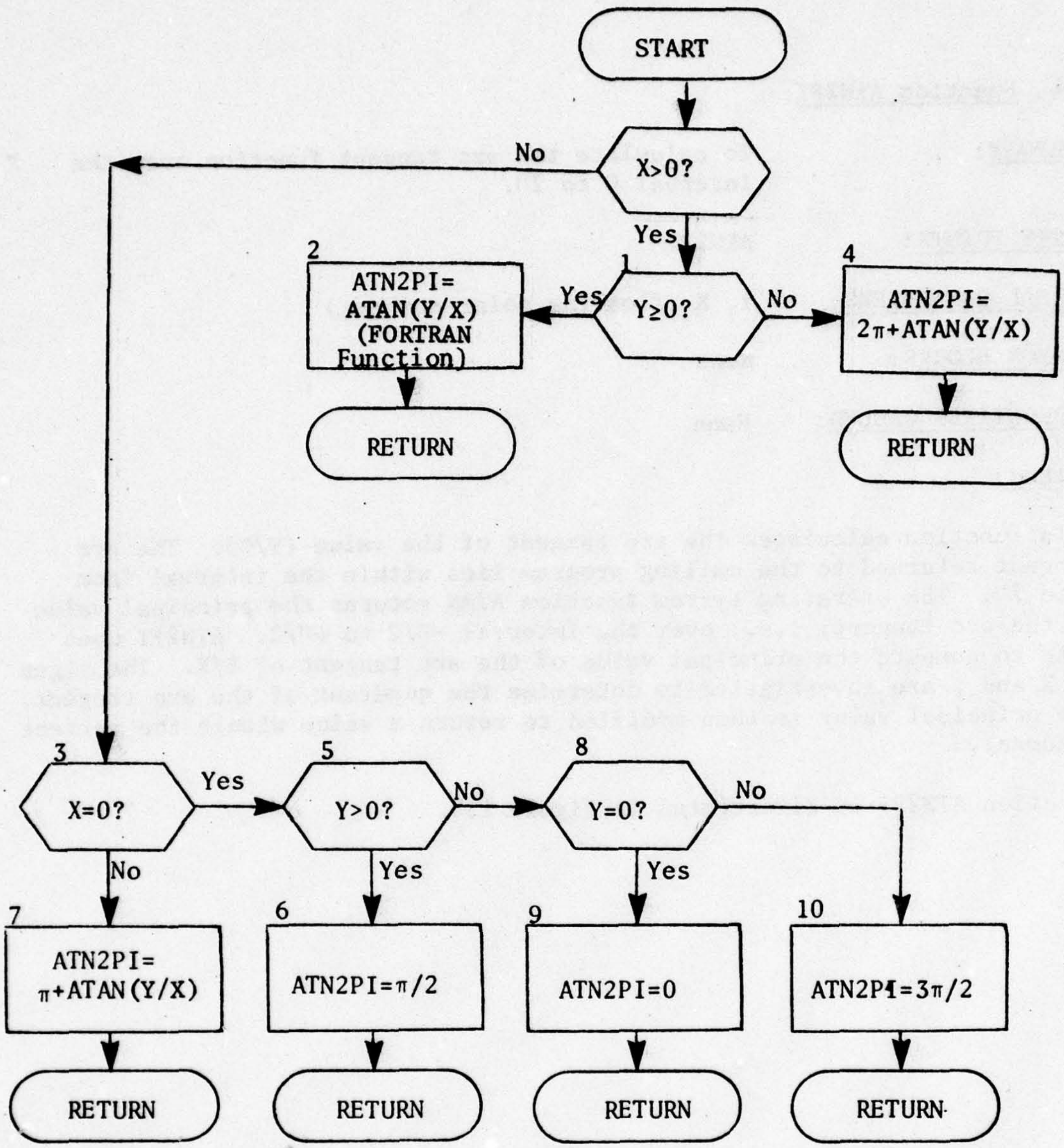


Figure 133. Function ATN2PI

9.5 Subroutine CINSGET

PURPOSE:	Call INSGET and update index
ENTRY POINTS:	CINSGET
FORMAL PARAMETERS:	L: Array for data from INSGET call LOC: Starting index N: Number of words from INSGET call LOCN: Ending index +1
COMMON BLOCKS:	None
SUBROUTINES CALLED:	INSGET

Method:

This subroutine calls INSGET in order to place N words into array L. LOC is the starting position into INSGETs arrays. LOCN is the next location into INSGET array's after CINSGET exits.

Subroutine CINSGET is illustrated in figure 134.

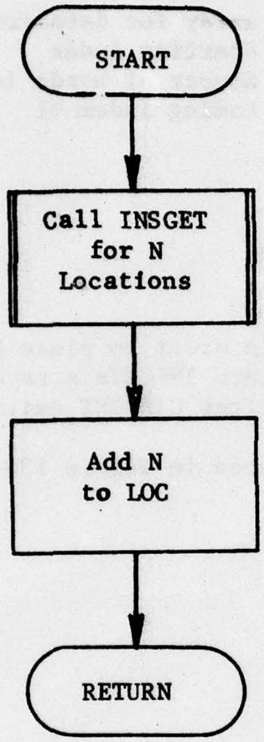


Figure 134. Subroutine CINSGET

9.6 Function DELLONG

PURPOSE: To compute the signed difference between two longitudes.

ENTRY POINTS: DELLONG

FORMAL PARAMETERS: A - A floating point longitude
B - A floating point longitude

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method:

The input longitudes are expressed as decimal degrees measured in a westerly direction. West longitudes are in the range 0-180 degrees; east longitudes are in the range 180-360 degrees. Longitudes 0 and 360 are the Greenwich meridian. This function returns the value of the difference between the two longitudes. The sign of the value is determined as follows:

Positive if $A \geq B$
Negative if $A < B$.

Function DELLONG is illustrated in figure 135.

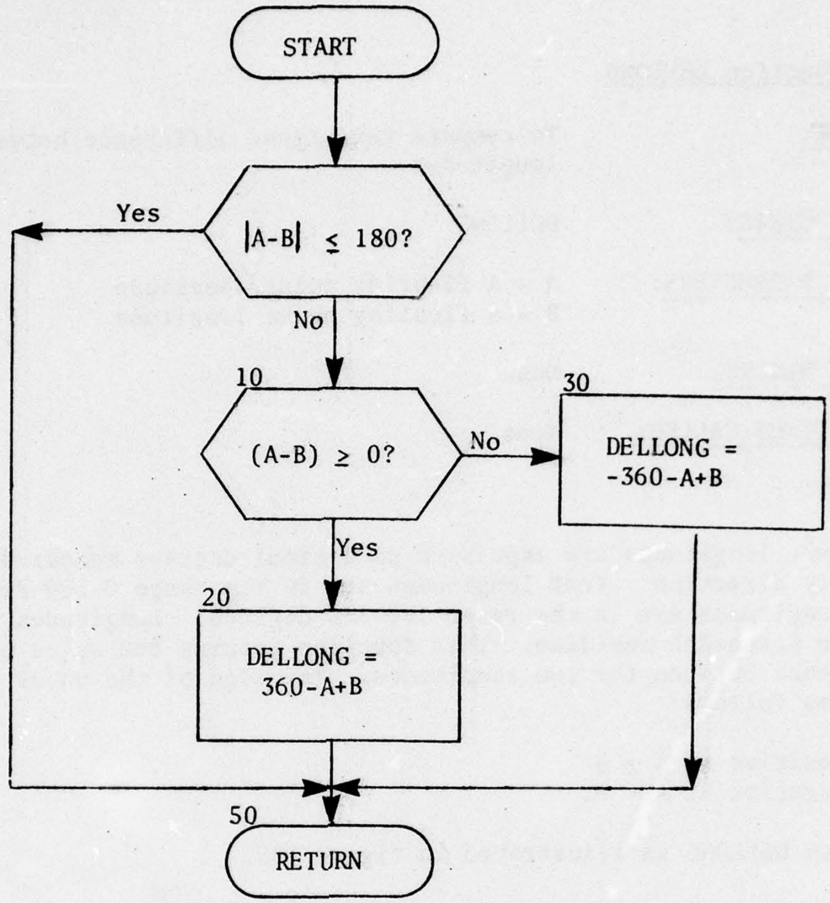


Figure 135. Function DELLONG

9.7 Function DIFFLONG

PURPOSE:	To compute the difference between two longitudes whose sign is determined by the shorter direction of travel from the first meridian to the second.
ENTRY POINTS:	DIFFLONG, DIFFLNG*
FORMAL PARAMETERS:	X1 - Floating point longitudes X2 - Floating point longitudes
COMMON BLOCKS:	None
SUBROUTINES CALLED:	None

Method:

The input longitudes lie in the interval 0-360 degrees with west longitudes in the range 0-180 degrees and east longitudes in the range 180-360 degrees. Longitudes 0 and 360 define the Greenwich meridian. This function returns a value whose absolute value is equal to the number of degrees of longitude traversed in using the shorter great circle route from meridian X1 to meridian X2. The sign is positive if the direction of travel is eastward and negative otherwise.

Function DIFFLONG is illustrated in figure 136.

* Duplicate entry for DIFFLONG.

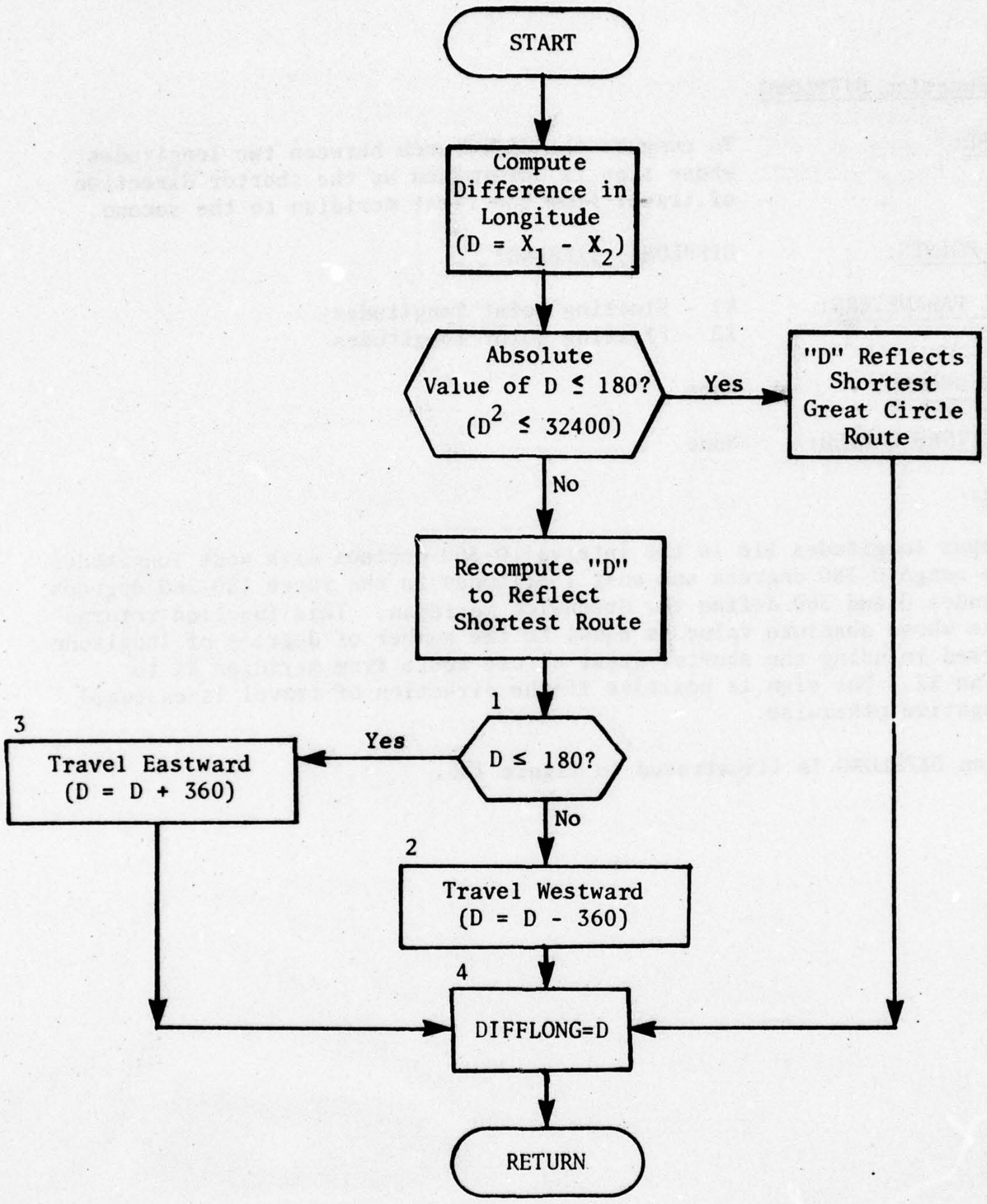


Figure 136. Function DIFFLONG

9.8 Function DISTF

PURPOSE: To compute great circle distances in nautical miles.

ENTRY POINTS: DISTF, DIST, DSTF

FORMAL PARAMETERS:

- LAT1 - Latitude of point 1
- LONG1 - Longitude of point 1
- LAT2 - Latitude of point 2
- LONG2 - Longitude of point 2
- DIN - Difference in Longitudes for Entry DSTF

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method:

The purpose of this function is to calculate the great circle distance between two points using the law of cosines for a spherical triangle.

There are three entry points to this function: DISTF, DSTF, and DIST. DISTF uses the latitudes and longitudes of the two points to calculate the distance. DSTF is the same as DISTF except that the difference in longitude is passed instead of the two longitudes. DIST, which is only called by EVALPLAN in EVALALOC, is the same as DISTF except that it returns the square of the distance.

The formal parameters are all type real. The coordinates are input in degrees with south latitude and east longitude coordinates being negative. The value returned is in nautical miles.

Function DISTF is illustrated in figure 137.

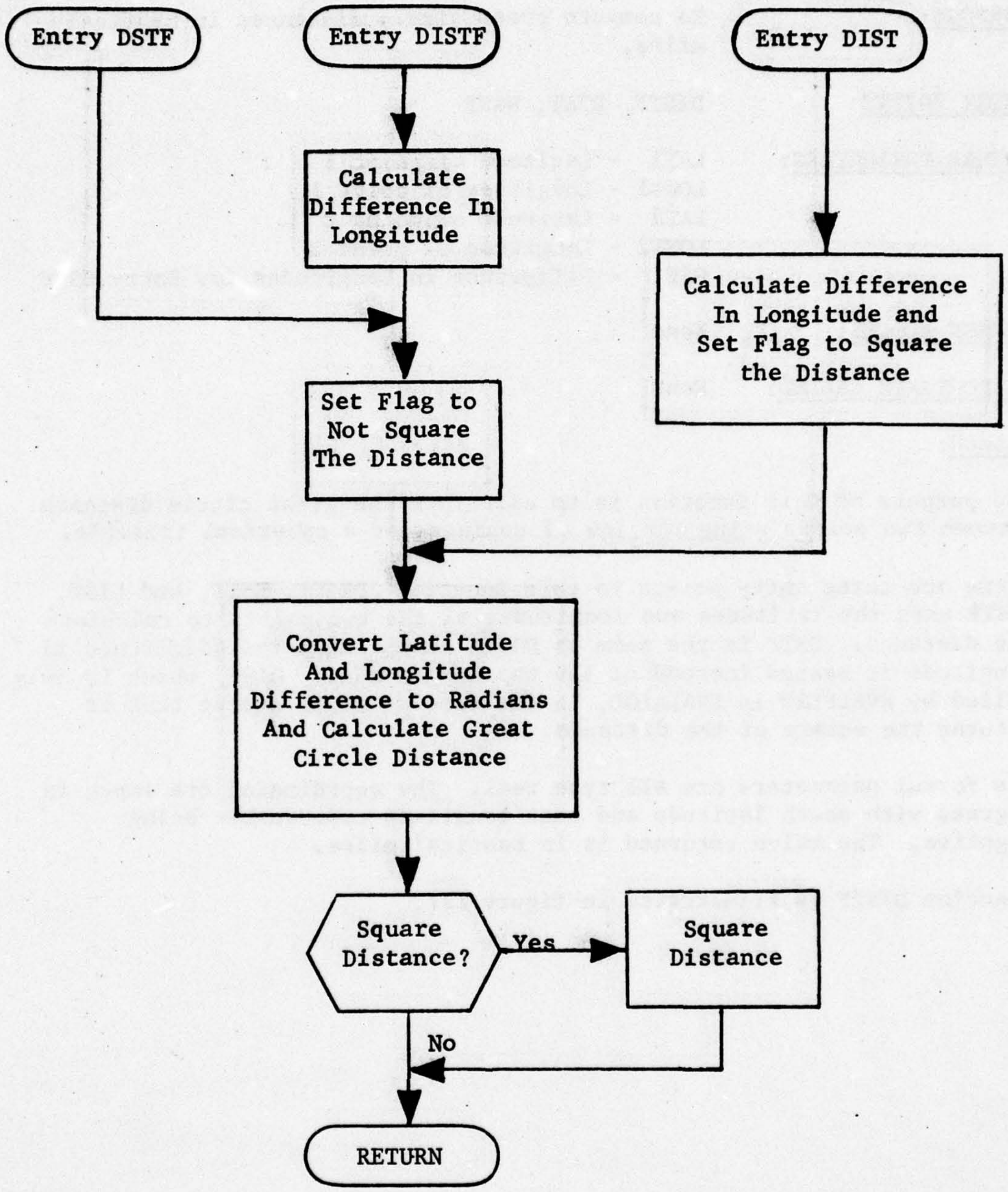


Figure 137. Function DISTF

9.9 Subroutine DOTLINE

PURPOSE: To plot a dotted line between two points.

ENTRY POINTS: DOTLINE

FORMAL PARAMETERS:
X1 - x-coordinate of first point
Y1 - y-coordinate of first point
X2 - x-coordinate of second point
Y2 - y-coordinate of second point

COMMON BLOCKS: None

SUBROUTINES CALLED: PLOT*

Method:

To draw a dotted line between two points, the pen has to plot and skip segments of the line alternately. The distance between the two points is computed and after each segment of plotted line, the distance traveled by the pen from the first point is compared with the length of the line. If the line is not yet completed, a constant segment of the line is skipped and the next segment plotted until the second point is reached.

Subroutine DOTLINE is illustrated in figure 138.

---

* This is a standard plotting routine and not documented in this manual. It is only included for completeness.

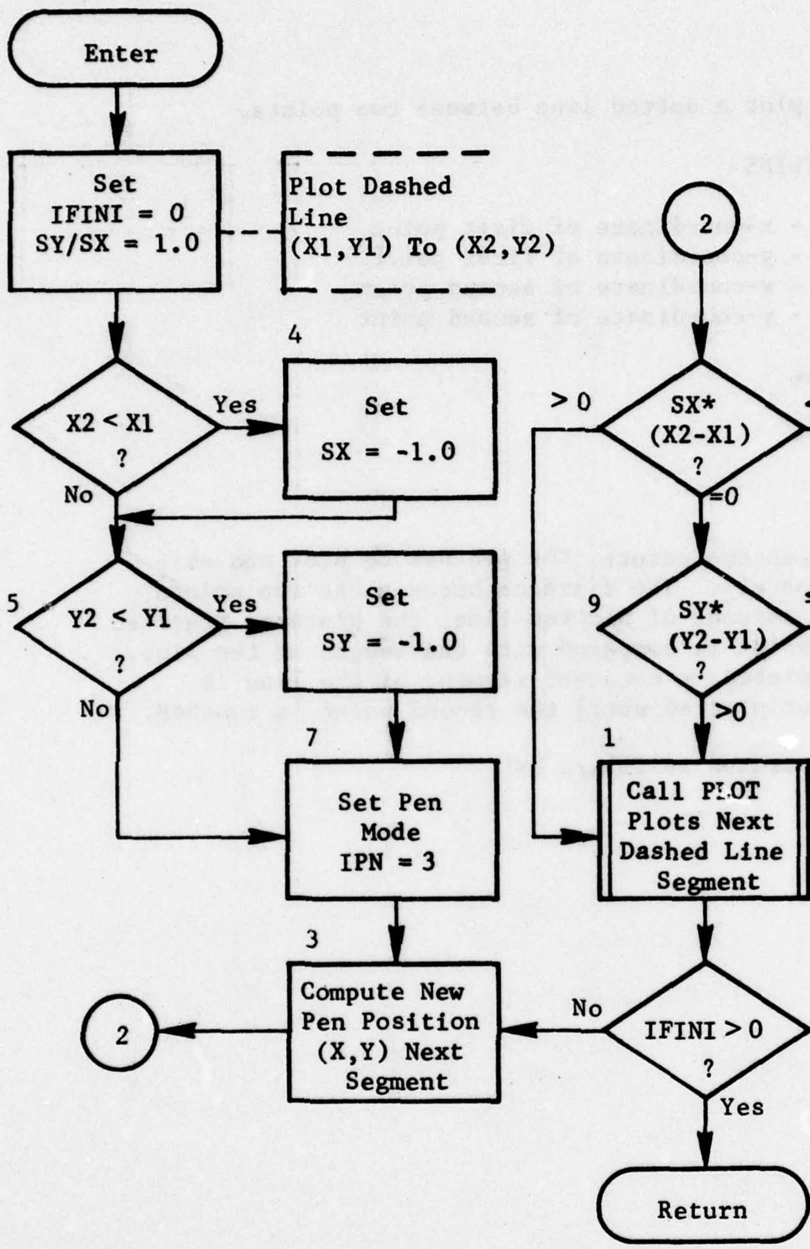


Figure 138. Subroutine DOTLINE

9.10 Subroutine FINDCLAS

PURPOSE: Finds a list of values for the attribute ACLASS

ENTRY POINTS: FINDCLAS

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C30, CLASSES

SUBROUTINES CALLED: HEAD, NEXTTT

Method:

This subroutine uses the directory portion of the data organization index. First the INDATR record for ACLASS is found. Then the VALIST chain is retrieved with each value contained thereon stored in common block CLASSES.

Subroutine FINDCLAS is illustrated in figure 139.

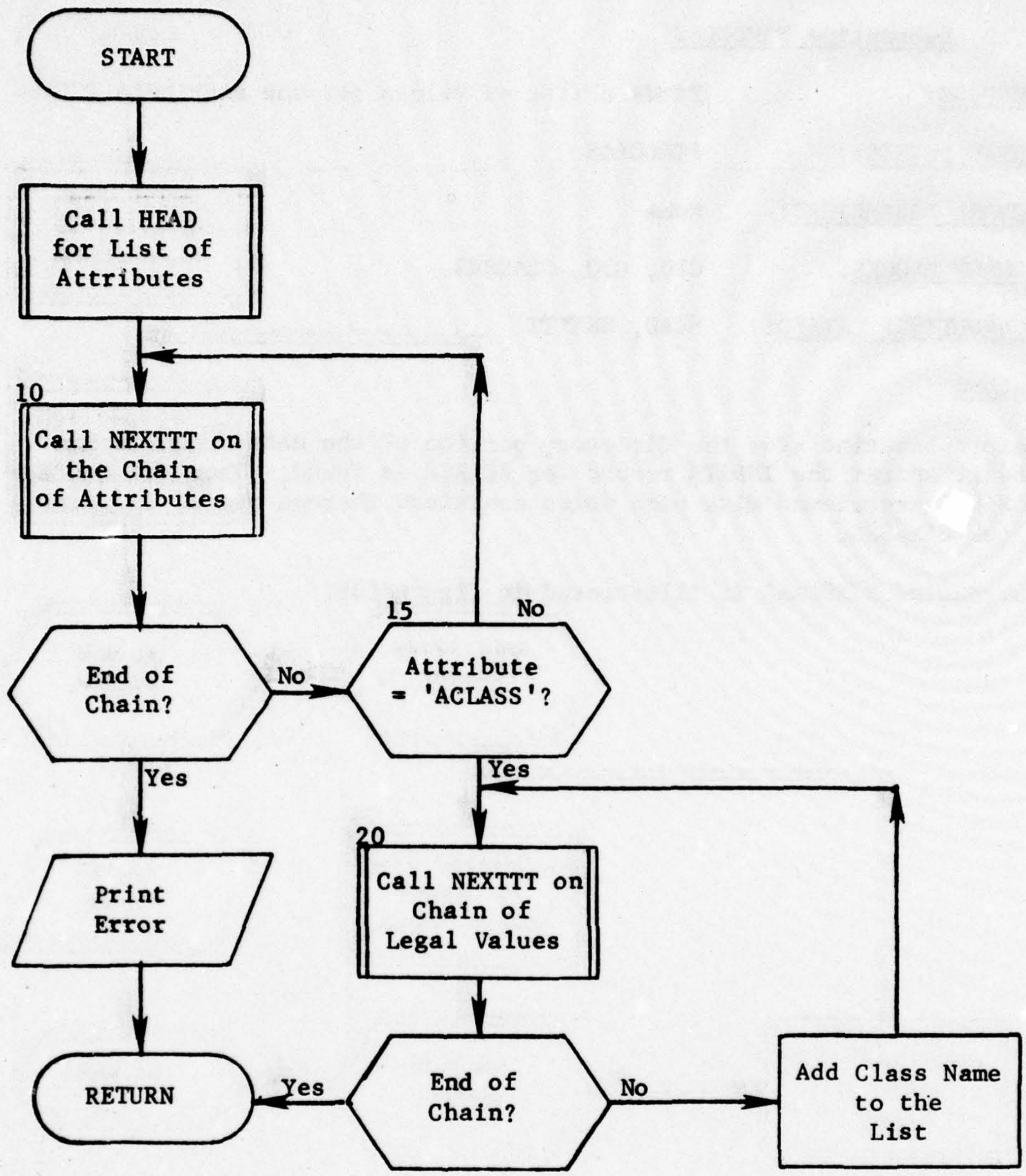


Figure 139. Subroutine FINDCLAS

9.11 Subroutine FINDSIDE

PURPOSE: Make a list of the valid values for attribute SIDE

ENTRY POINTS: FINDSIDE

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C25, SIDES

SUBROUTINES CALLED: HEAD, NEXTTT

Method:

This subroutine uses the NAMEZ chain of the data organization index. By retrieving the entire chain all values for SIDE that have been used to create headers are found. The unique values are stored in common block SIDES.

Subroutine FINDSIDE is illustrated in figure 140.

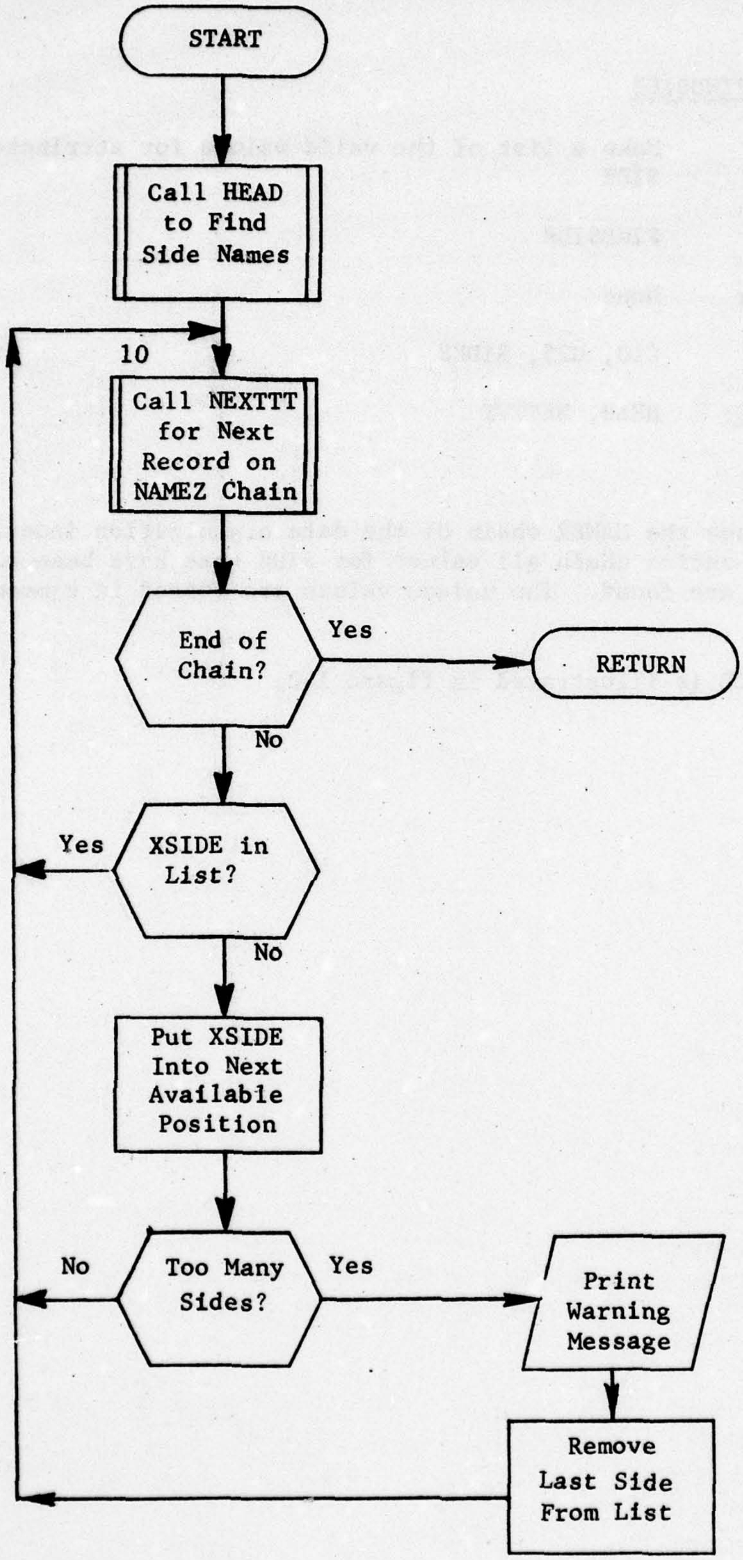


Figure 140. Subroutine FINDSIDE

9.12 <u>Subroutine FORMAK</u>

<u>PURPOSE</u>:	Inserts character strings into array FORMAR
<u>ENTRY POINTS</u>:	FORMAK
<u>FORMAL PARAMETERS</u>:	STRING: String of characters for insert NUMBER: Number of characters in string
<u>COMMON BLOCKS</u>:	DSPFRM
<u>SUBROUTINES CALLED</u>:	None
<u>CALLED BY</u>:	BLDOTH, DSPMAK

<u>Method</u>:

First NUMBER is checked. If it is zero this is an initializing call. The FORMAR array is blanked out. IFPNT (the FORMAR character bit pointer) is set to zero and IFLNG (the FORMAR word pointer) is set to one.

If NUMBER is nonzero, ISPT (the STRING character bit pointer) is set to zero and ISLN (the STRING word pointer) is set to one. Then the following process takes place NUMBER times. The STRING character defined by ISPT and ISLN is stored in the FORMAR character defined by IFPNT and IFLNG. Both ISPT and IFPNT are incremented by 6. If either exceed 30, it is reset to 0 and its corresponding word pointer is incremented.

Subroutine FORMAK is illustrated in figure 141.

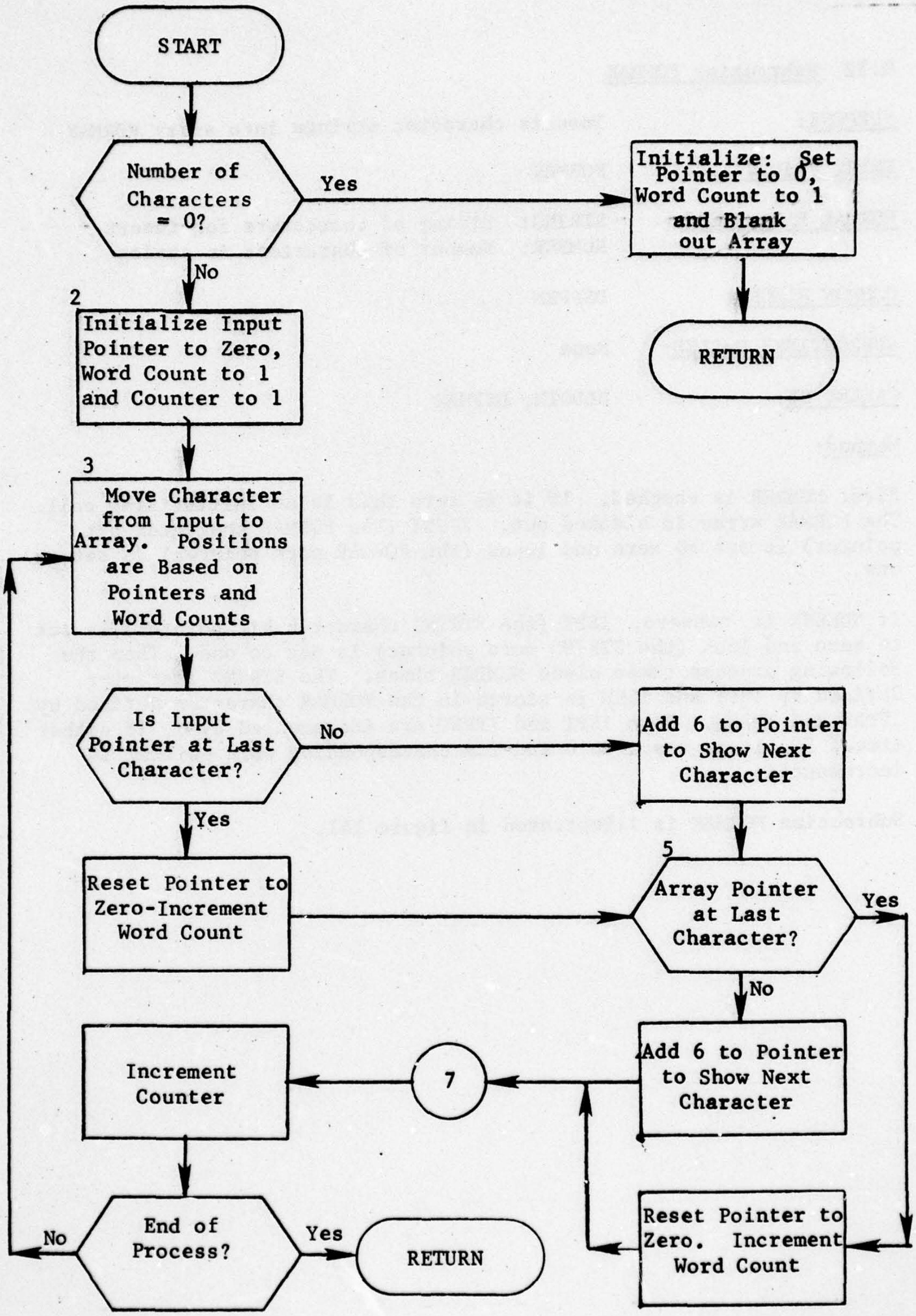


Figure 141. Subroutine FORMAK

9.13 Function GETCLOCK

PURPOSE: To return the current time in floating point minutes.

ENTRY POINTS: GETCLOCK, GETCLK (duplicate entry)

FORMAL PARAMETERS: X - A dummy parameter

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method:

This assembly language subroutine is used to return the current time in floating point minutes.

Function GETCLOCK is illustrated in figure 142.

AD-A054 310 COMMAND AND CONTROL TECHNICAL CENTER WASHINGTON D C F/G 15/7
THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK) PROGR--ETC(U)
JUN 77 D J SANDERS, P F MAYKRANTZ, J M HERRIN

UNCLASSIFIED CCTC-CSM-MM-9-77-V1-PT-2 SBIE-AD-E100 052 NL

4 OF 6
AD
A054310

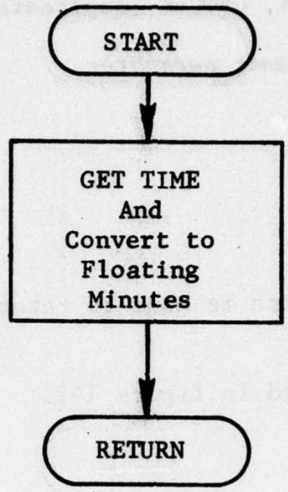


Figure 142. Function GETCLOCK

9.14 Function GETDATE

PURPOSE: To obtain the current date.

ENTRY POINTS: GETDATE

FORMAL PARAMETERS: X - A dummy parameter

COMMON BLOCKS: None

SUBROUTINES CALLED: DATIM

Method:

This function calls the system function DATIM to obtain the current date in the format MM/DD/YY. When called in a FORTRAN program, this function will return the date in a character *6 (MMDDYY).

Function GETDATE is illustrated in figure 143.

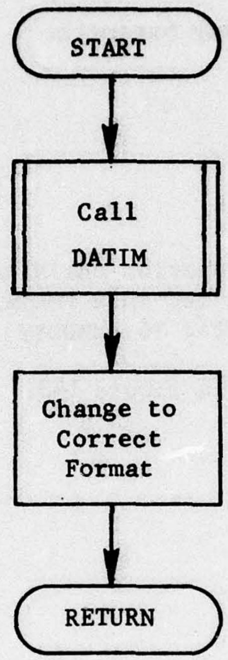


Figure 143. Function GETDATE

9.15 Subroutine GETNXT

PURPOSE:	Execute a retrieval scheme
ENTRY POINTS:	GETNXT
FORMAL PARAMETERS:	BRANCH: 1=normal return 2=end of scheme
COMMON BLOCKS:	C10, C15, C20, C25, C30, SCHEME
SUBROUTINES CALLED:	HEAD, NEXTTT, RETRV

Method:

This subroutine executes the retrieval scheme in common block SCHEME. Its execution presupposes that:

1. A retrieval scheme has been stored in SCHEME

2. The INDRCT record has been retrieved for the record type of the primary header

3. The variable POINT has been set to the next scheme instruction to be executed (originally POINT=1)

GETNXT begins by setting BRANCH=1 and looking at the value in SCHEME indexed by POINT. The next step depends upon the value of SCHEME (POINT).

SCHEME(POINT)=1 (Get Header)

This process depends in part on the value of SCHEME(POINT+1) which indicates whether there are values for the CLASS and SIDE attributes. In any case, NEXTTT is called on the MYNAMZ chain until either a record is retrieved which satisfies any value for CLASS and SIDE or the MYNAMZ chain master is found. In the first case, the identified header is retrieved and POINT set to the next instruction. If the master is found, the scheme is complete. BRANCH is set 2 and the subroutine exits.

SCHEME(POINT)=2 (Get Next for Chain)

NEXTTT is called for the chain named in SCHEME(POINT+1). If the record type retrieved is the chain's master (SCHEME(POINT+2)), POINT is set equal to SCHEME(POINT+3) which is the index of the previous Get Next or Get Header instruction. Otherwise POINT is set to the next instruction.

SCHEME(POINT)=3 (Head Chain)

HEAD is called for the chain named in SCHEME(POINT+1) and POINT is set to the next instruction.

SCHEME(POINT)=4 (Return)

POINT is set to SCHEME(POINT+1) which is the previous Get Next or Get Header instruction and the subroutine exits having retrieved a complete record set.

Subroutine GETNXT is illustrated in figure 144.

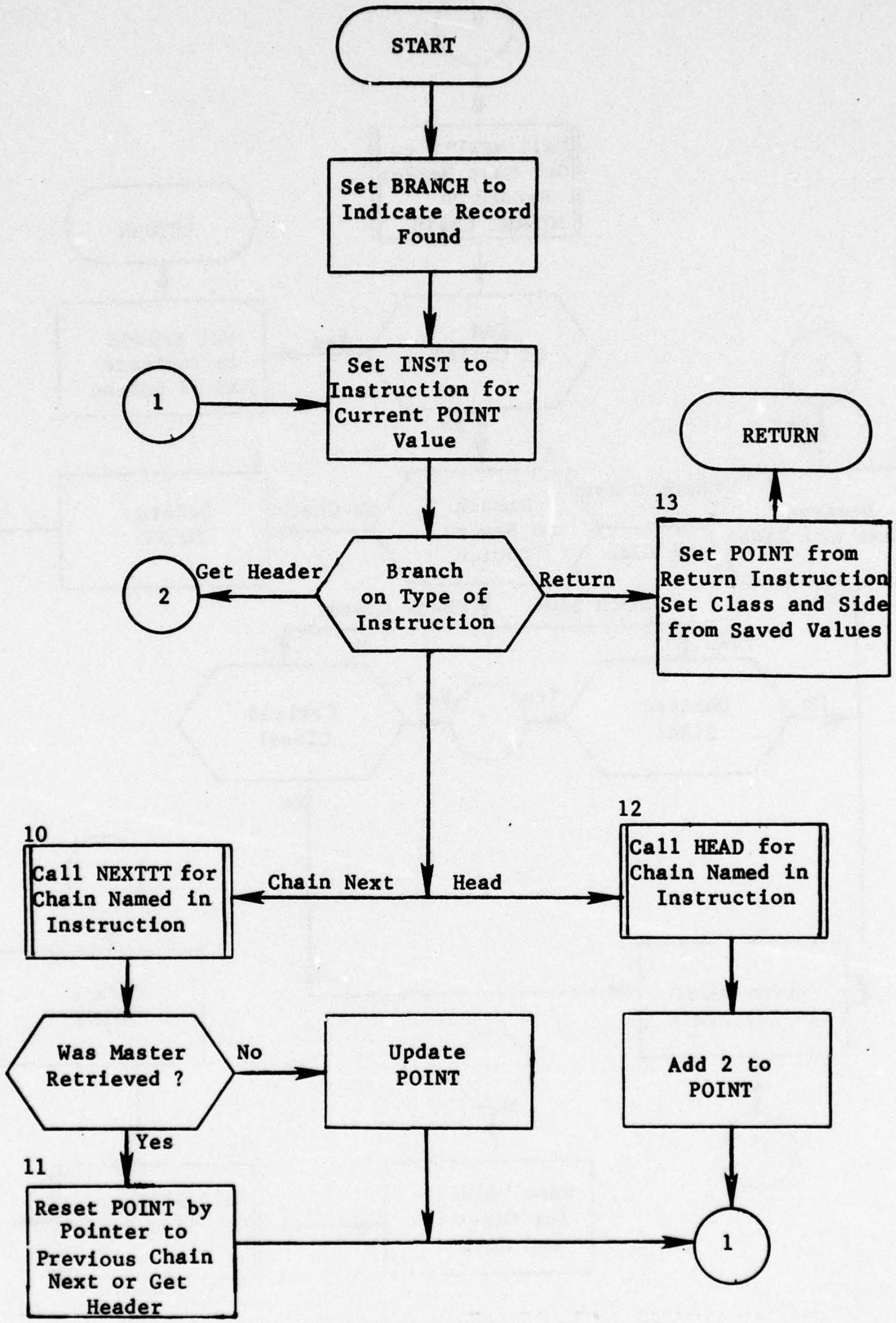


Figure 144. Subroutine GETNXT (Part 1 of 2)

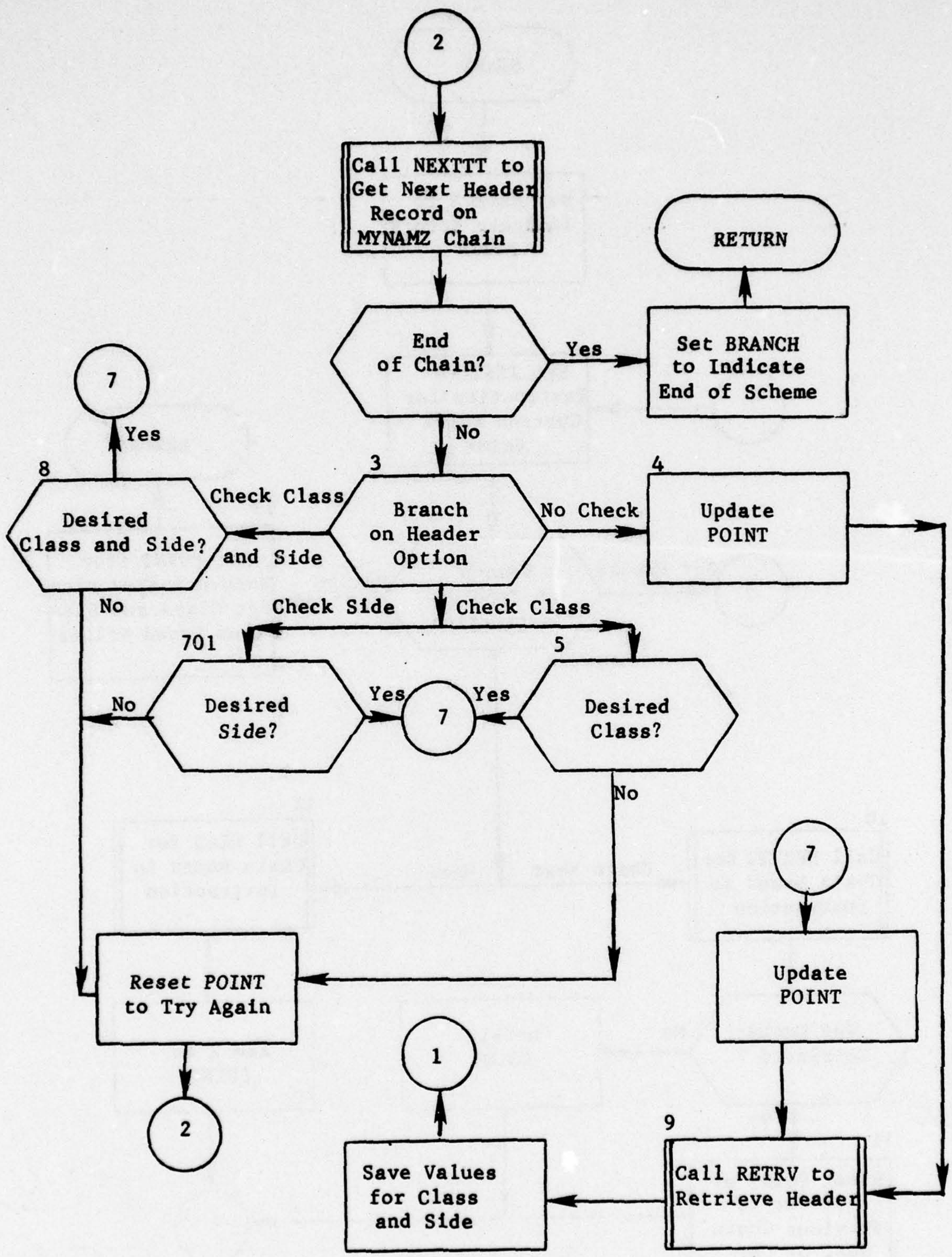


Figure 144. (Part 2 of 2)

9.16 Subroutine GETSTR*

PURPOSE: To obtain the next input string and make first interpretation

ENTRY POINTS: GETSTR

FORMAL PARAMETERS: None

COMMON BLOCKS: IPQT, STRING

SUBROUTINES CALLED: None

CALLED BY: ERRFND, INICOP, SRM

Method:

This subroutine which is a character-by-character analysis of an input string from a card image is best understood by reference to figure 145. Basically, the process attempts to discriminate between alphabetic and numeric data. Special characters which appear in array OPRATR also are discriminated. A string is terminated by a blank or a member of OPRATR.

---

*Note that this subroutine resides in the Main overlay of COP.

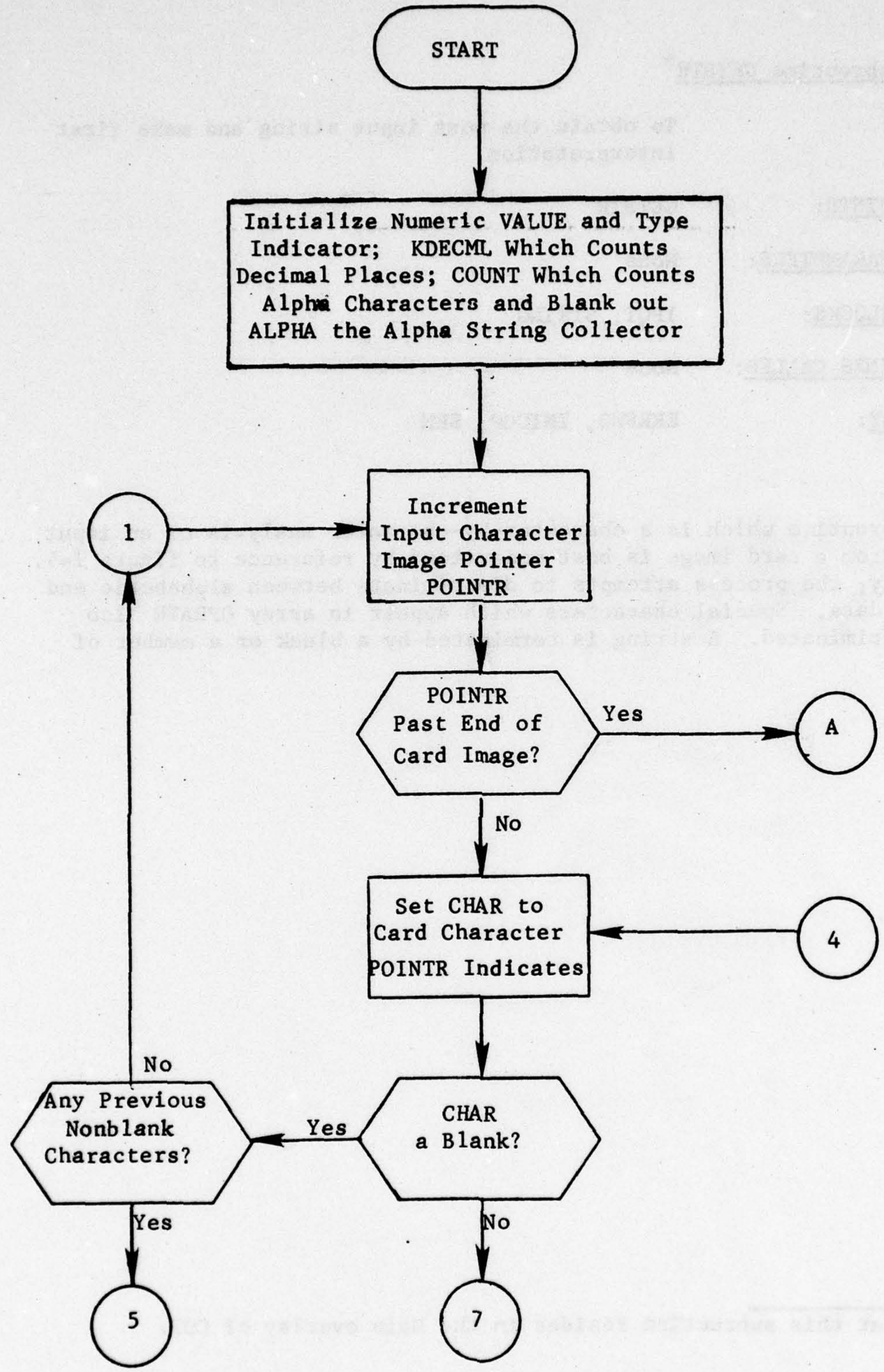


Figure 145. Subroutine GETSTR (Part 1 of 6)

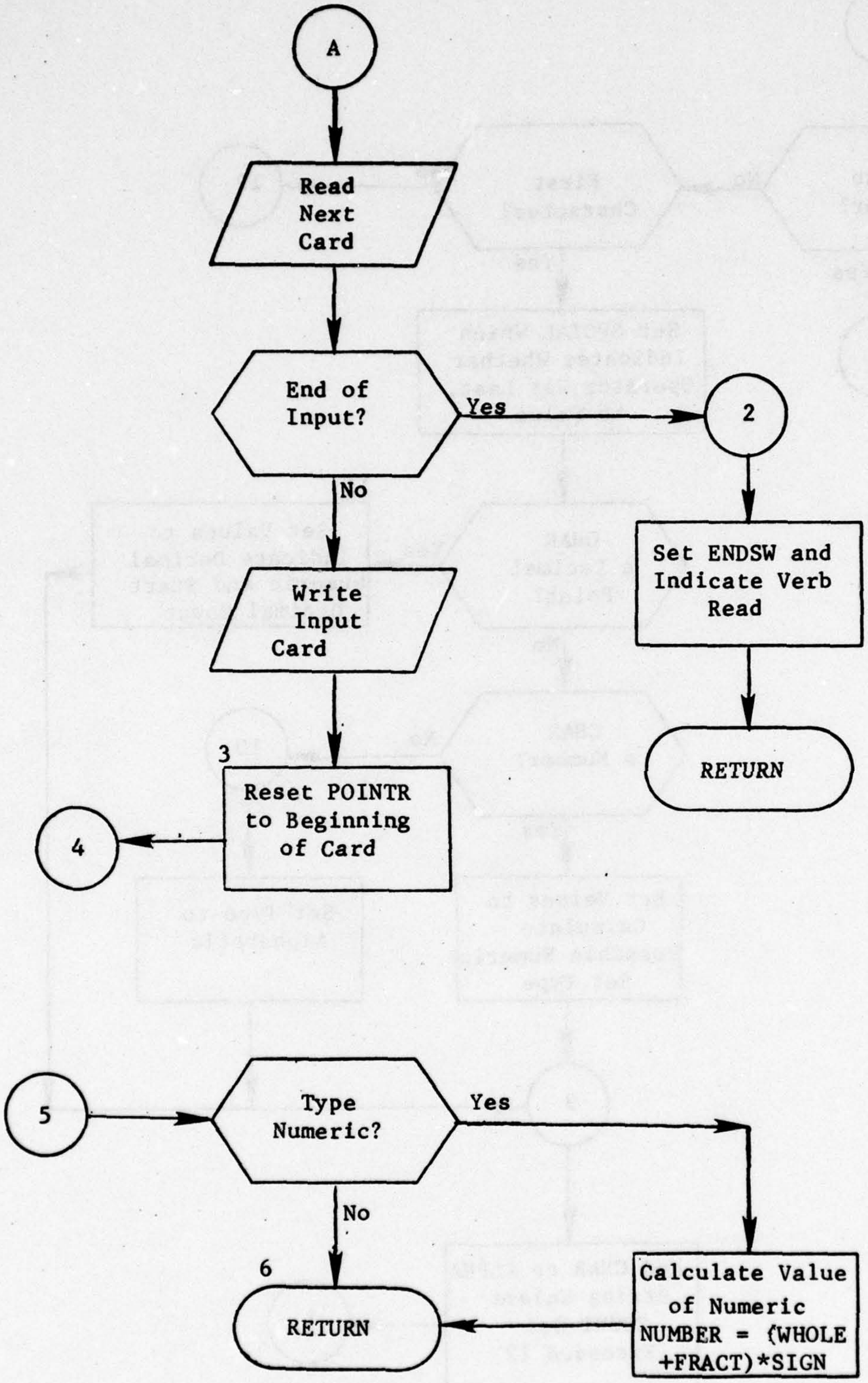


Figure 145. (Part 2 of 6)

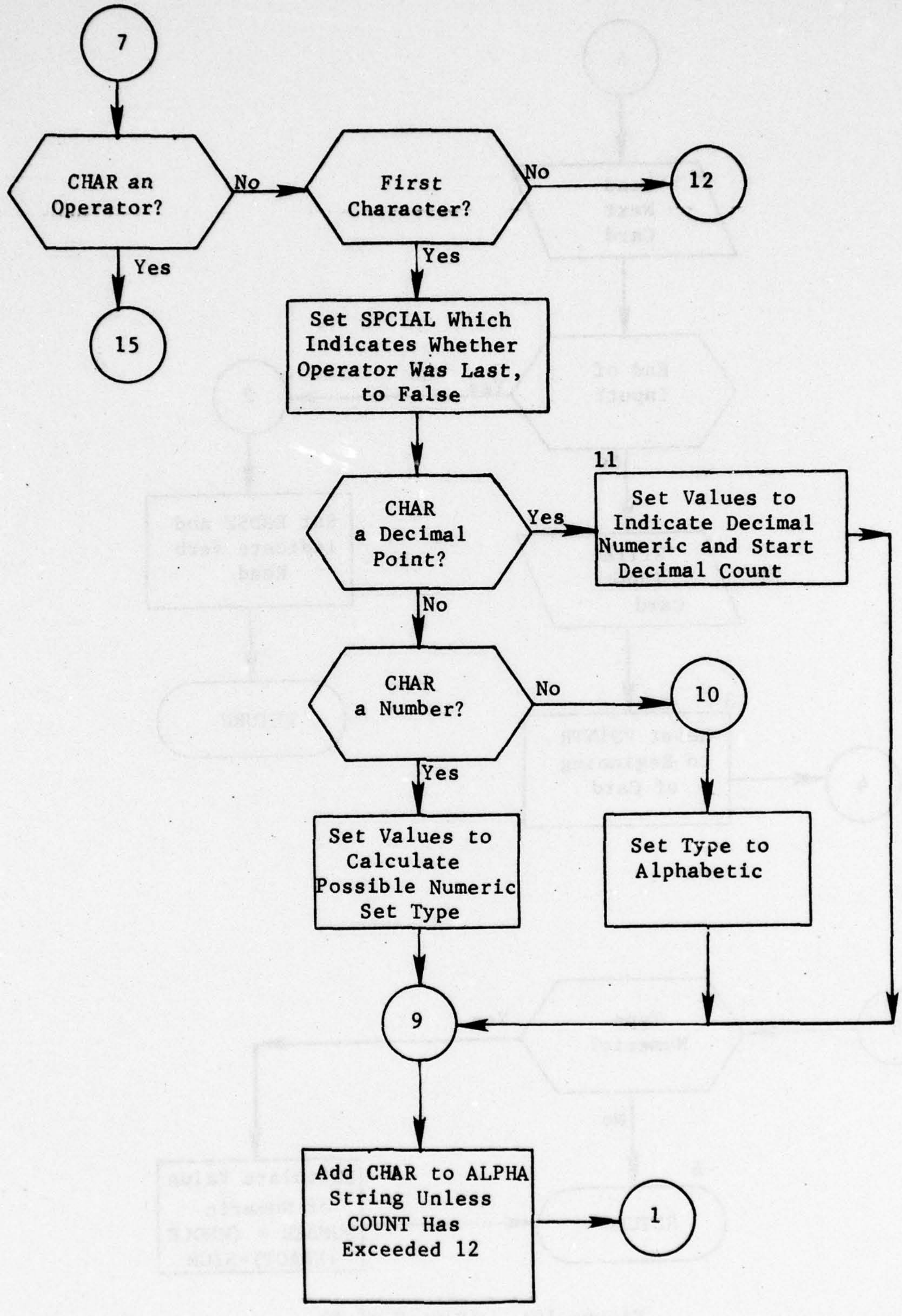


Figure 145. (Part 3 of 6)

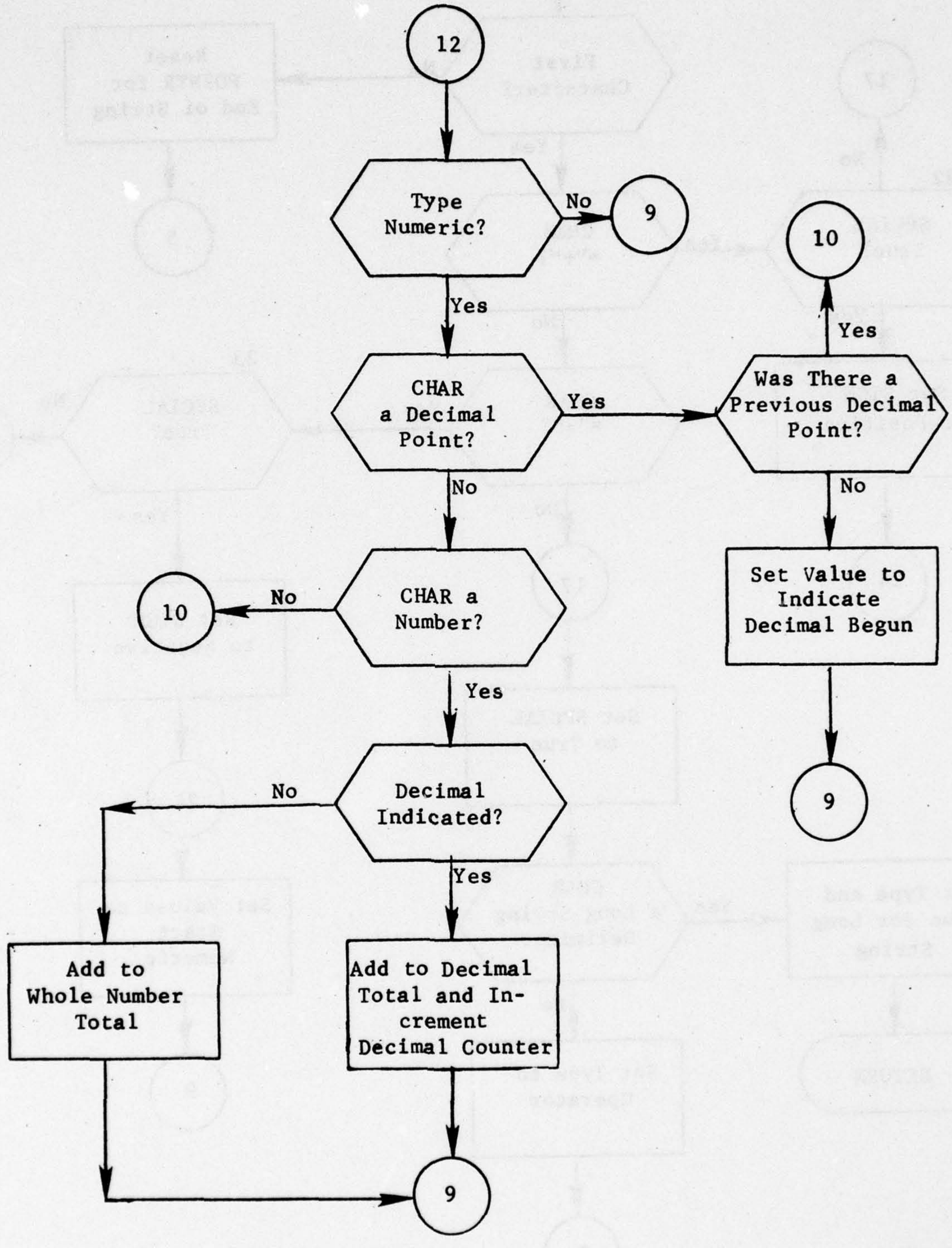


Figure 145. (Part 4 of 6)

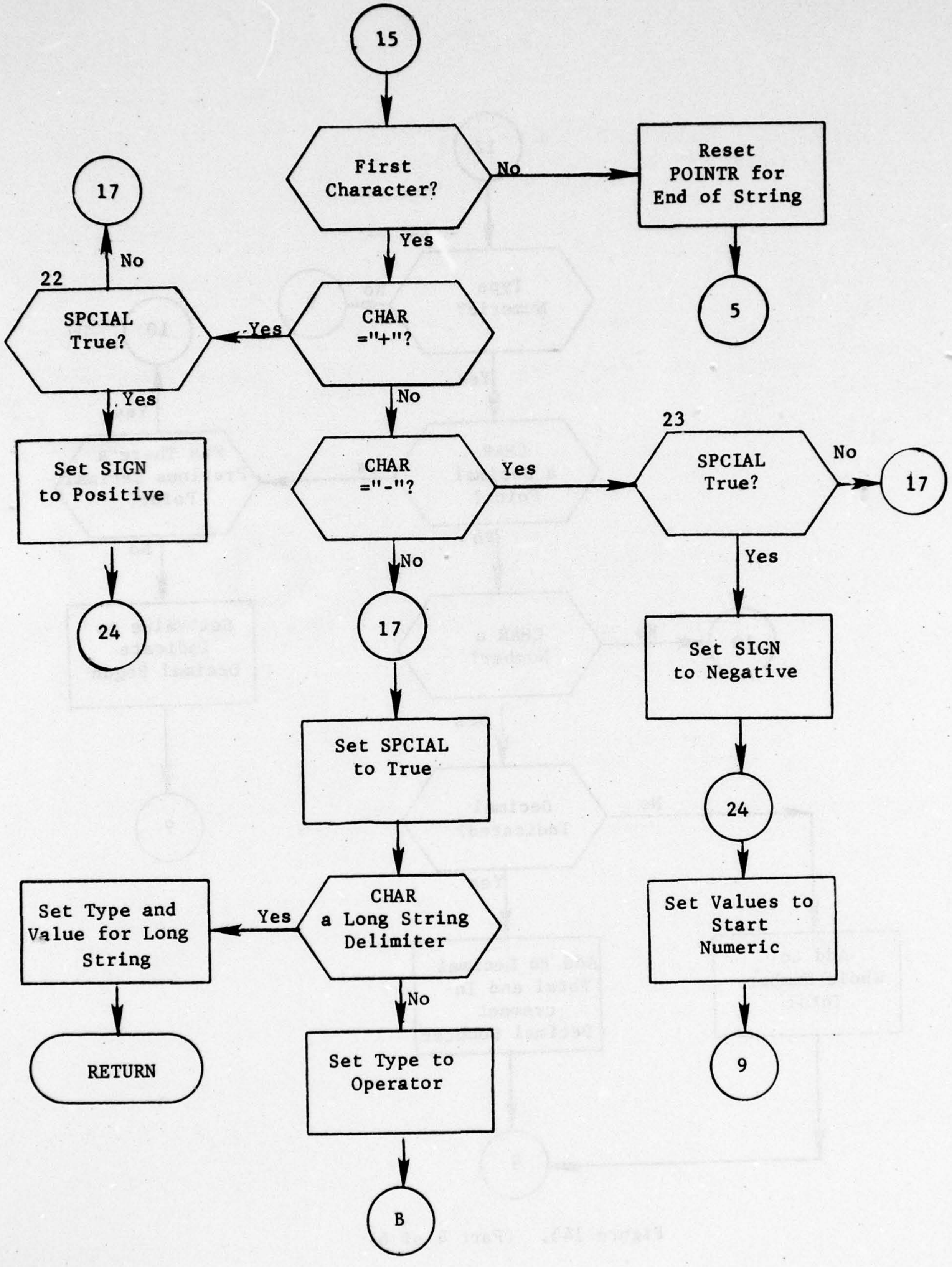


Figure 145. (Part 5 of 6)

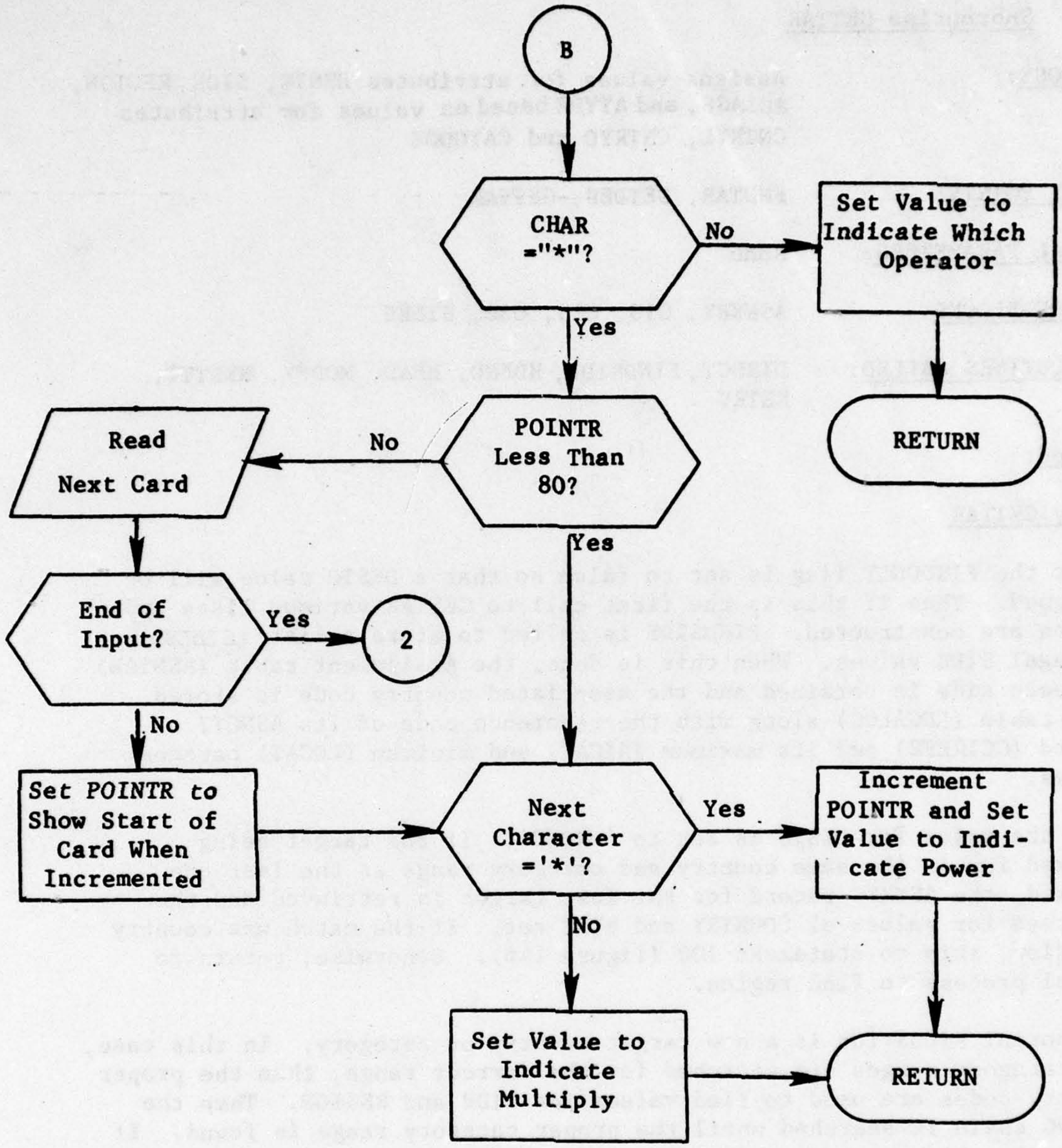


Figure 145. (Part 6 of 6)

9.17 Subroutine GETTAR

PURPOSE: Assigns values for attributes DESIG, SIDE, REGION, ACLASS, and ATYPE based on values for attributes CNTRYL, CNTRYO and CATCODE

ENTRY POINTS: FNDTAR, GETDES, GETTAR

FORMAL PARAMETERS: None

COMMON BLOCKS: ASNKEY, C10, C15, C30, SIDES

SUBROUTINES CALLED: DIRECT, FINDSIDE, HDFND, HEAD, MODFY, NEXTTT, RETRV

Method:

Entry GETTAR

First the FINDONLY flag is set to false so that a DESIG value will be assigned. Then if this is the first call to GETTAR various lists and tables are constructed. FINDSIDE is called to store a list (SIDES) of legal SIDE values. When this is done, the Assignment table (ASNTAB) for each side is obtained and the associated country code is stored in a table (LEGALCC) along with the reference code of its ASNCTY record (CCIREFZ) and its maximum (HICAT) and minimum (LOCAT) category ranges.

Next the value for DESIG is set to 'WRONG'. If the target being checked for is the same country and category range as the last one checked, the ASNREC record for the last target is retrieved and the searched for values of COUNTRY and SIDE set. If the match was country location, skip to statement 100 (figure 146). Otherwise, return to normal process to find region.

The normal situation is a new target country or category. In this case, the category ranges are searched for the correct range, then the proper country codes are used to find values for SIDE and REGION. Then the ASNRNG chain is searched until the proper category range is found. If necessary, both country codes are used in this search. Now the ASNCTY record corresponding to the country code which resulted in the match is retrieved.

Statement 100 (figure 146)

The reference code of the ASNREC record, the country code, the category range, side and region are all saved so that processing of many targets in the same category and country will be more efficient. The TYPRNG

chain is now headed and the reference code of the ASNTYP record retrieved is saved in ASNKEY. The ALTYPE chain is also headed and a check made for a legal region. If call was to FNDTAR, the subroutine exits.

<u>Statement 110 (figure 146)</u>

The ALTDES chain is now searched for a DESIG alpha that is available. If none is found 'ZZ999' is assigned. Otherwise, a DESIG assignment is made and the Assignment table updated appropriately.

<u>Entry FNDTAR</u>

The FINDONLY switch is set to true and the GETTAR process followed.

<u>Entry GETDES</u>

ASNKEY is used to retrieve the current ASNTYP record and control passed to Statement 110 (figure 146).

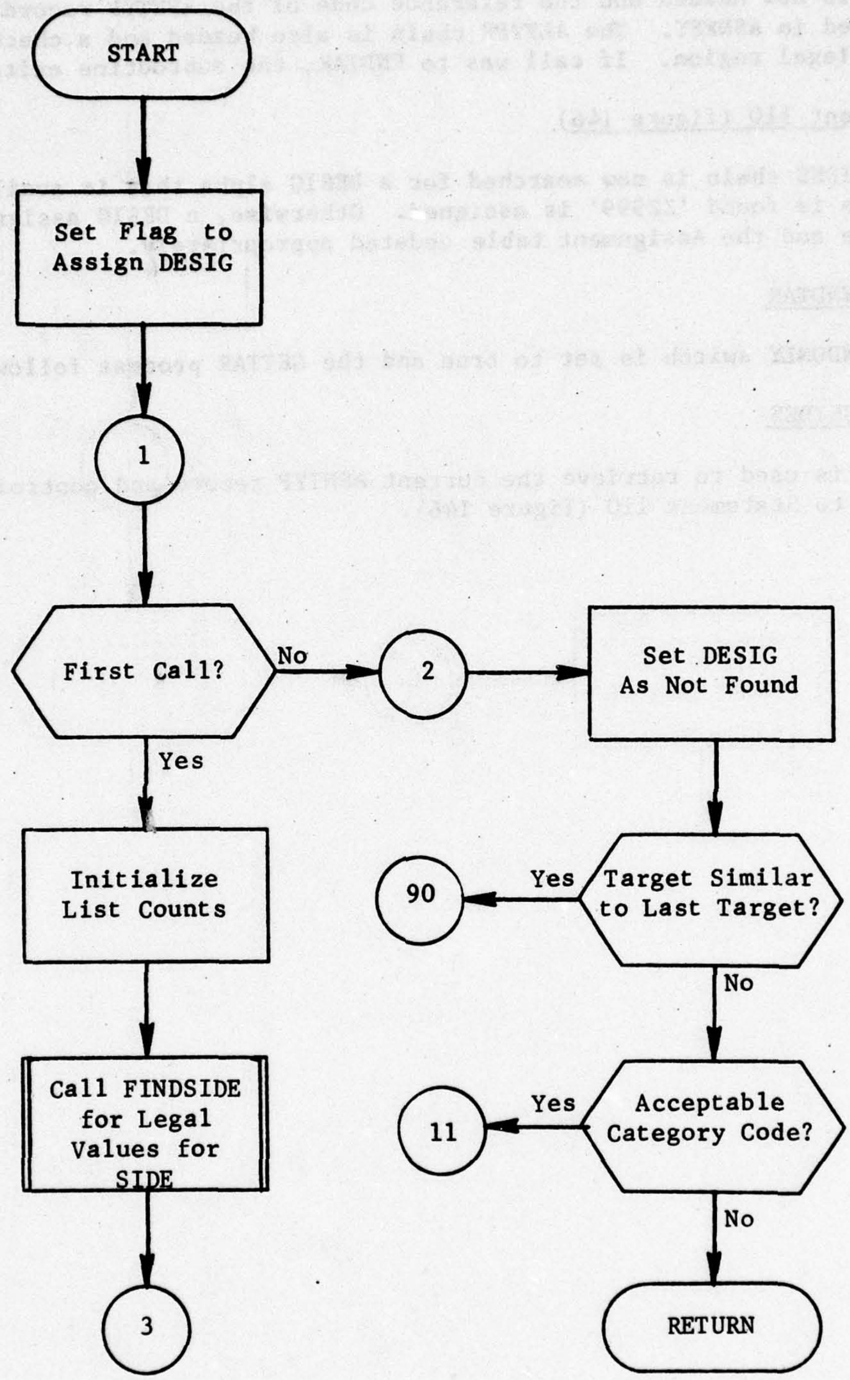


Figure 146. Subroutine GETTAR: Entry GETTAR (Part 1 of 9)

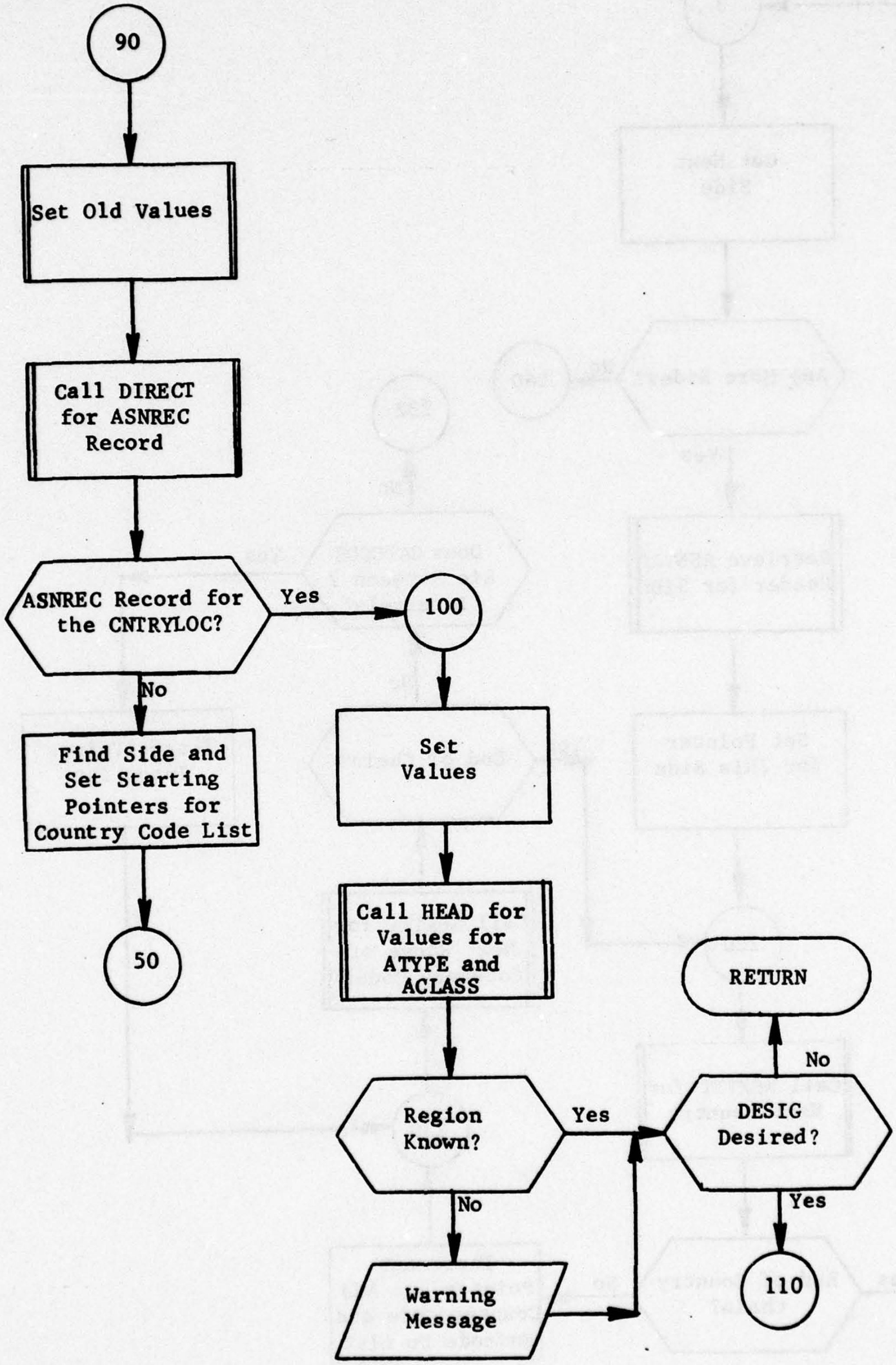


Figure 146. (Part 2 of 9)

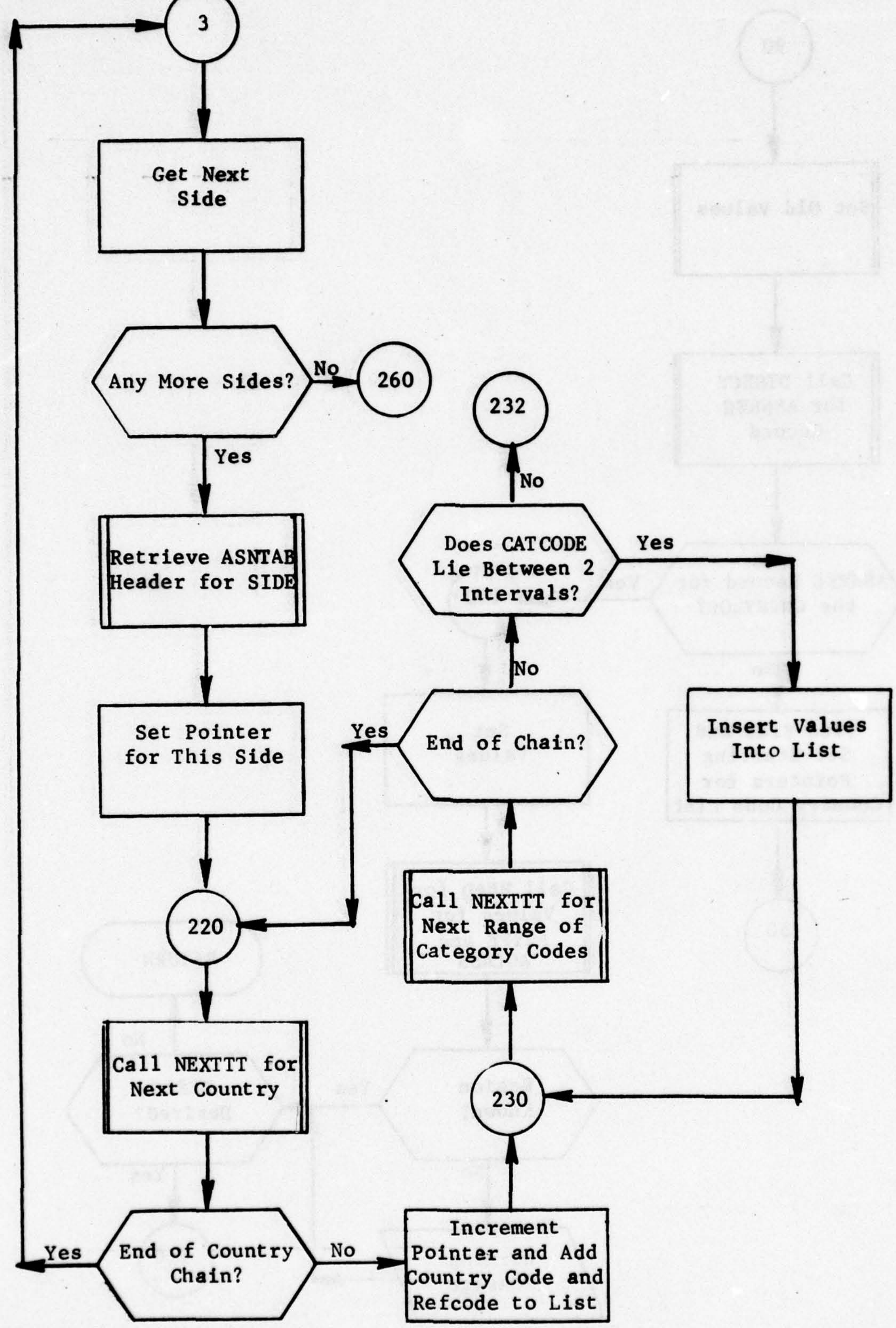


Figure 146. (Part 3 of 9)

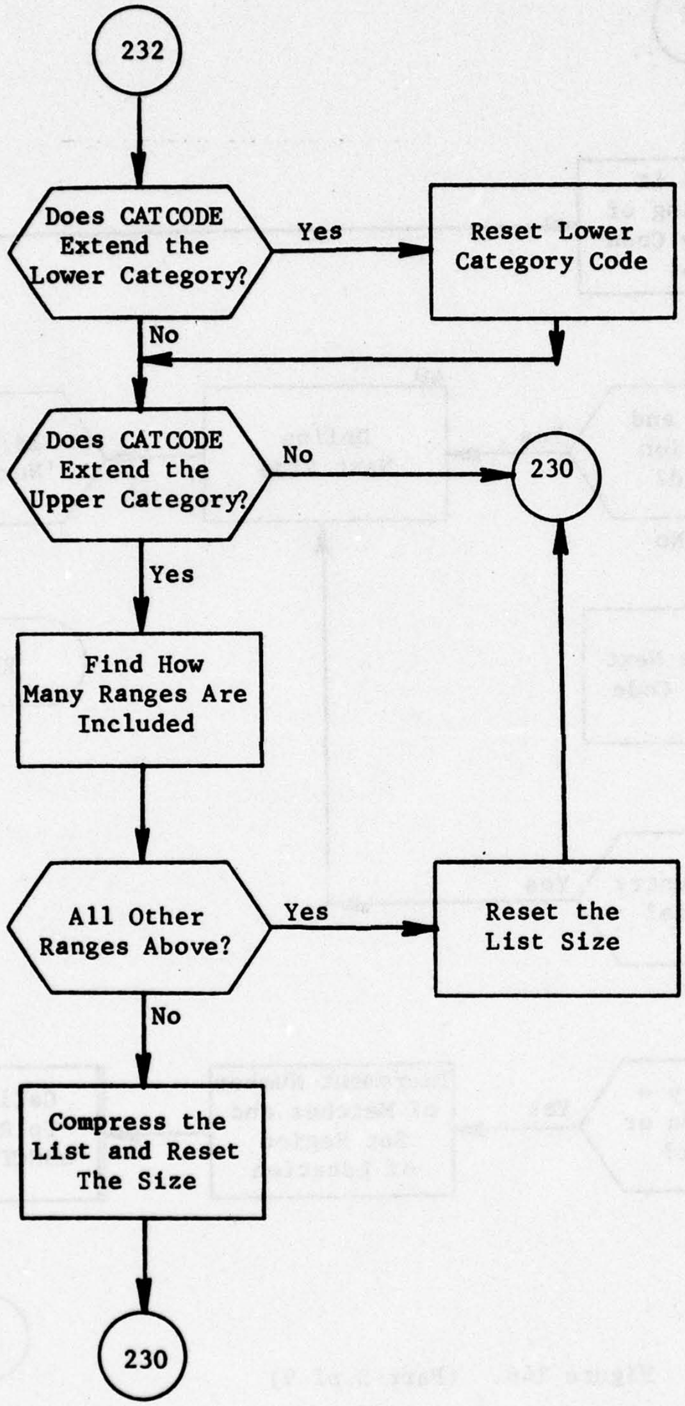


Figure 146. (Part 4 of 9)

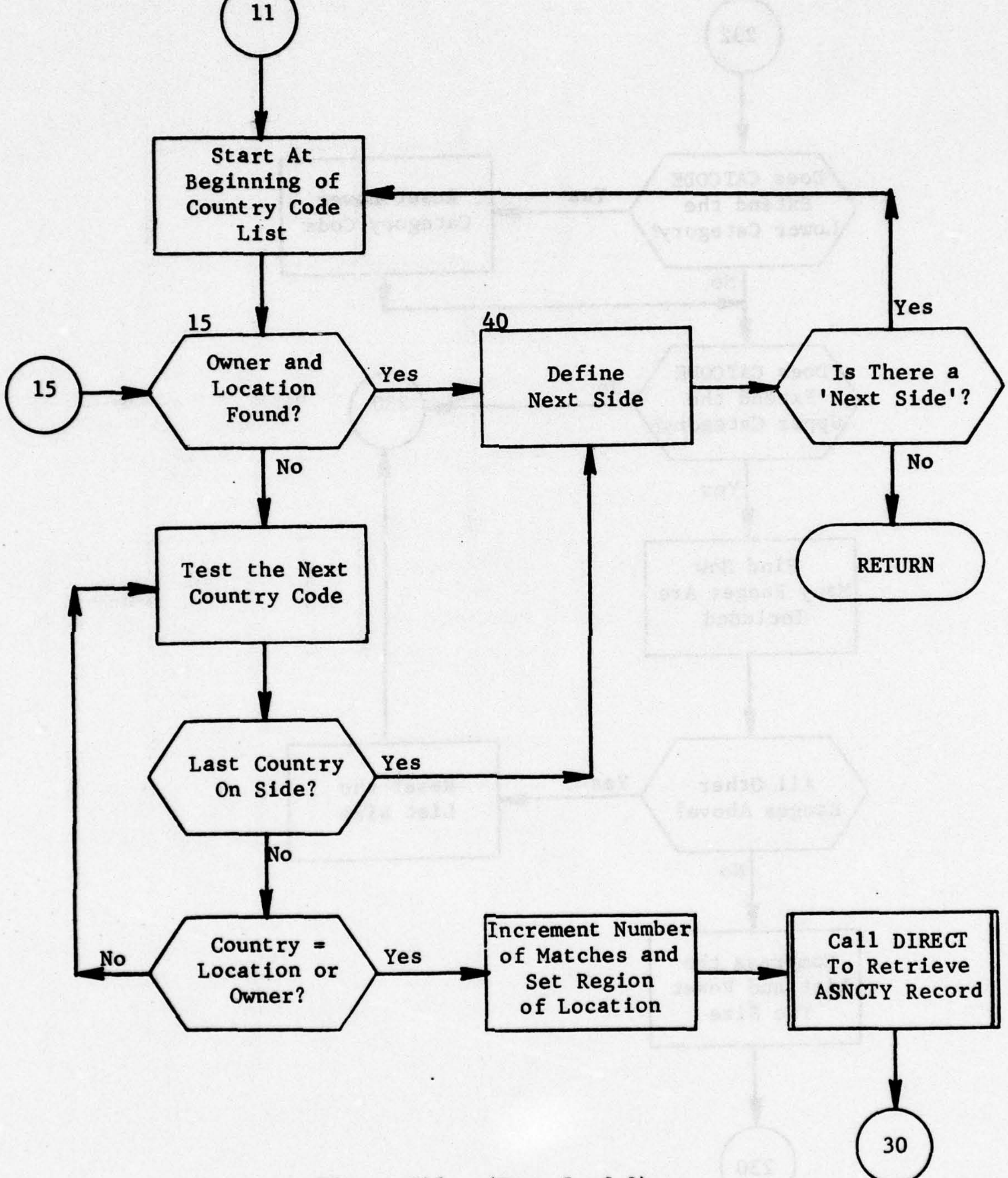


Figure 146. (Part 5 of 9)

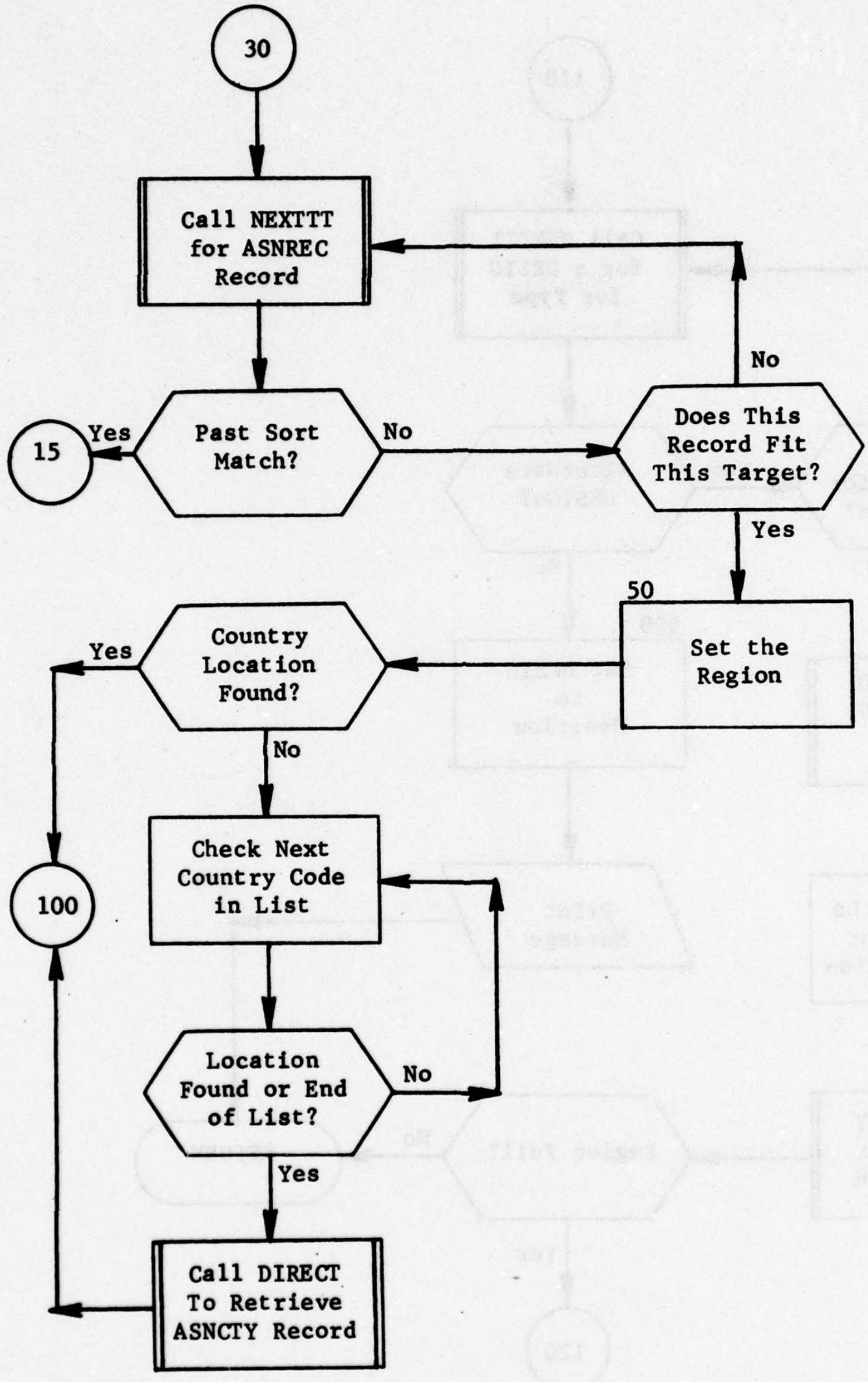


Figure 146. (Part 6 of 9)

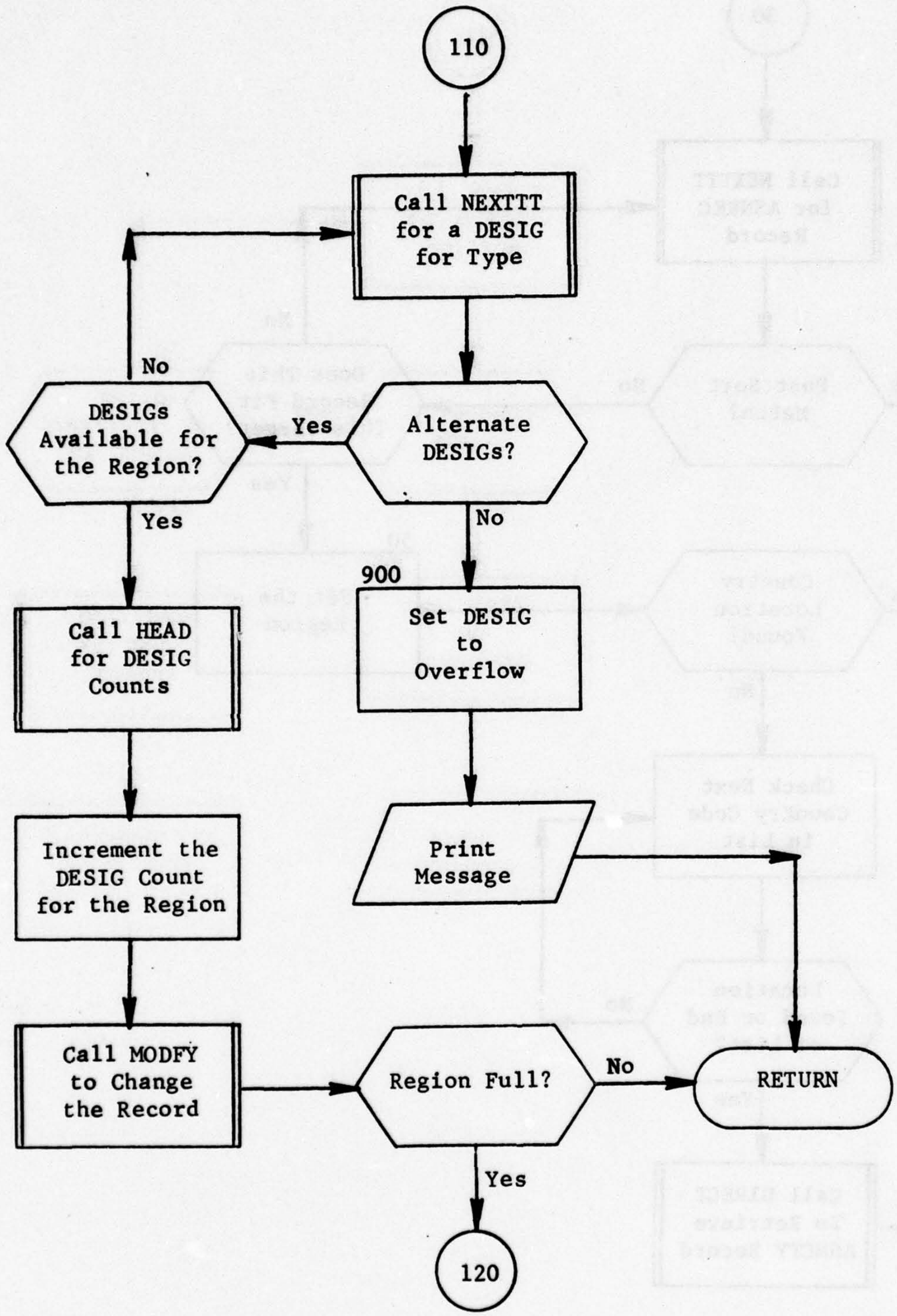


Figure 146. (Part 7 of 9)

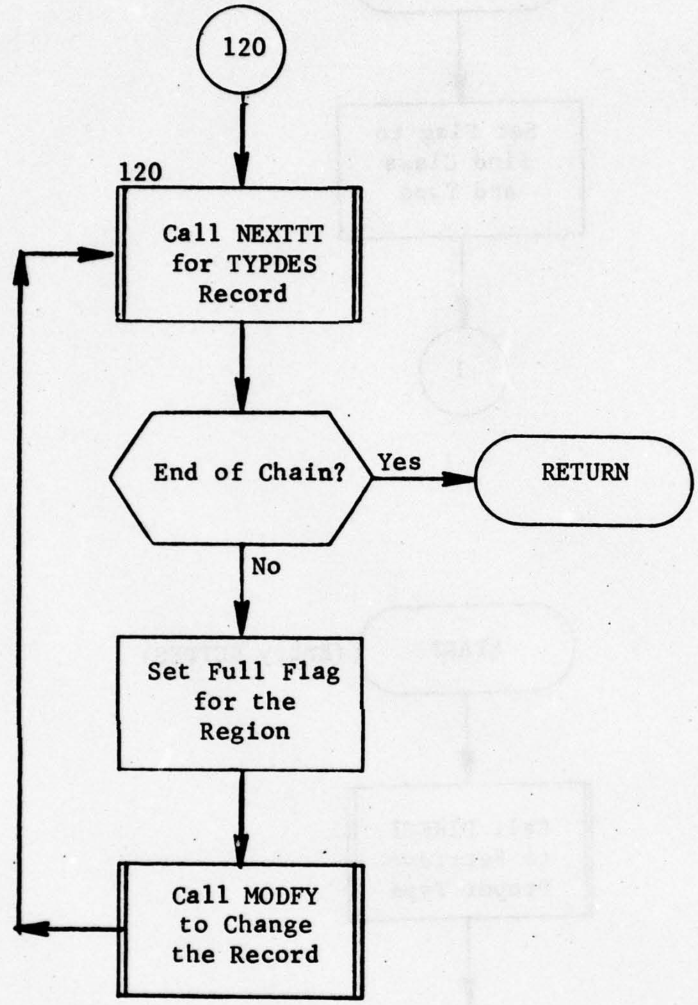


Figure 146. (Part 8 of 9)

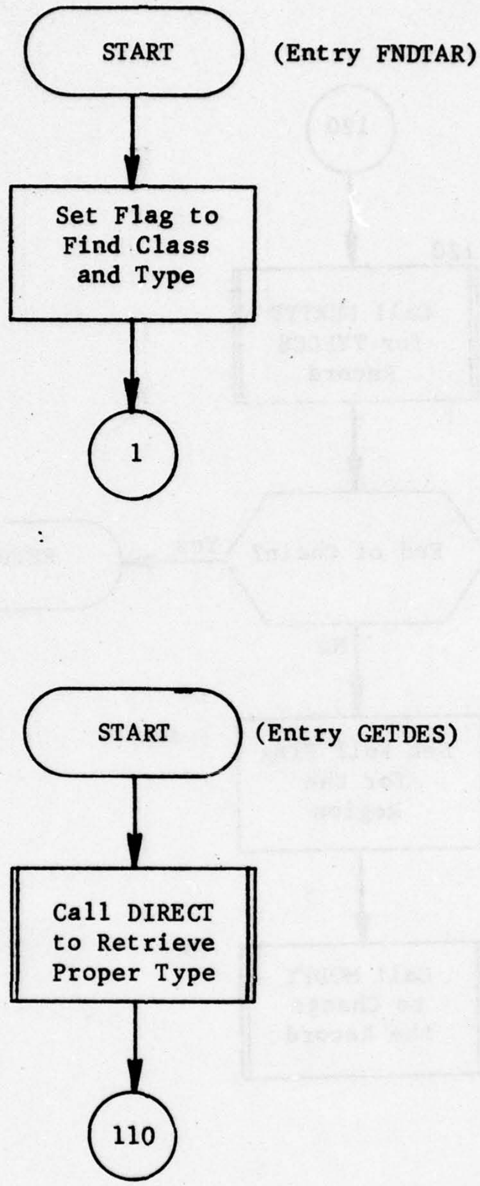


Figure 146. Subroutine GETTAR: Entries FNDTAR and GETDES
(Part 9 of 9)

9.18 Function GLOG

PURPOSE: This routine returns values (.T. = True; .F. = False) of QUICK system packed logical arrays which have been previously set by SLOG.

ENTRY POINTS: GLOG

FORMAL PARAMETERS:

L	- QUICK logical array
I, J, K	- Array indices
LL, M, N	- Array dimensions

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method:

The array designated by L is accessed. The position of the desired packed logical value is determined and the value extracted (1 = True; 0 = False) and the appropriate logical value returned in GLOG.

Function GLOG is illustrated in figure 147.

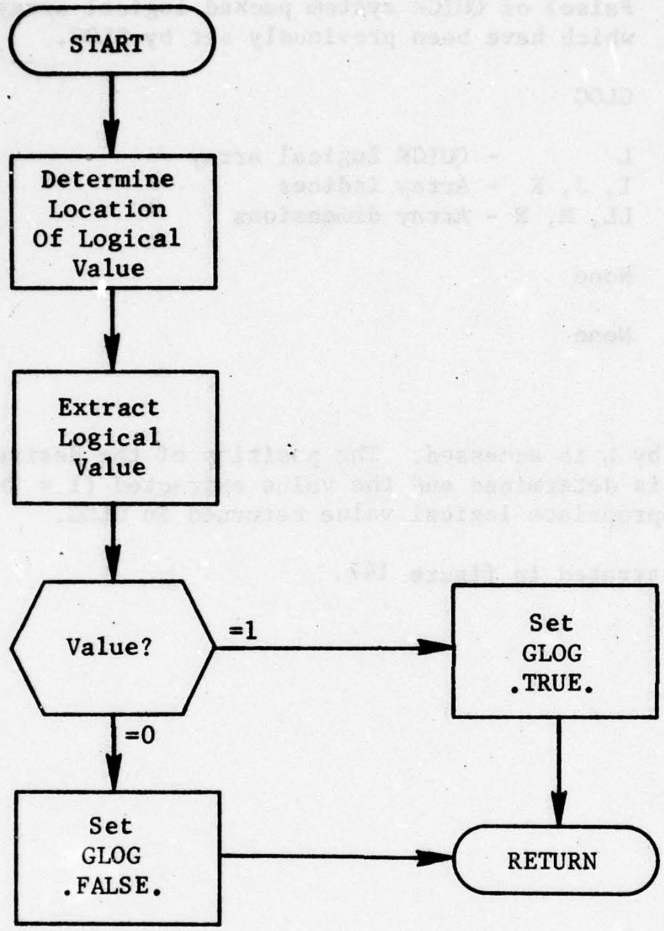


Figure 147. Function GLOG

9.19 Subroutine HOUSKEEP

PURPOSE:	To initialize the plot tape and position the pen at the end of each plot according to the size of the plot so that, if feasible, more than one plot may be on a single piece of paper.
ENTRY POINTS:	HOUSKEEP, HUSKP2, HUSKP3
FORMAL PARAMETERS:	None
COMMON BLOCKS:	PLTSPEC, TAPES
SUBROUTINES CALLED:	DOTLINE, NEWPEN*, PLOT*, PLOTS*, NUMBER*, SYMBOL*

Method:

One of the parameters the user can choose is the size of the plot and, therefore, the number of plots per page. This is indicated in the variable ISZE. Subroutine HOUSKEEP tests this indicator and sets XAXLEN and YAXLEN to the length of the x- and y-axes for the desired plot size. Then it positions the pen at the origin of the plot. At entry HUSKP2, the color of the pen is changed and subroutine DOTLINE is called to outline the plot. At entry HUSKP3, the graph number is printed, the plot is ended and the plot number is incremented.

Subroutine HOUSKEEP is illustrated in figure 148.

* These are standard plotting routines and not documented in this manual. They are only included for completeness.

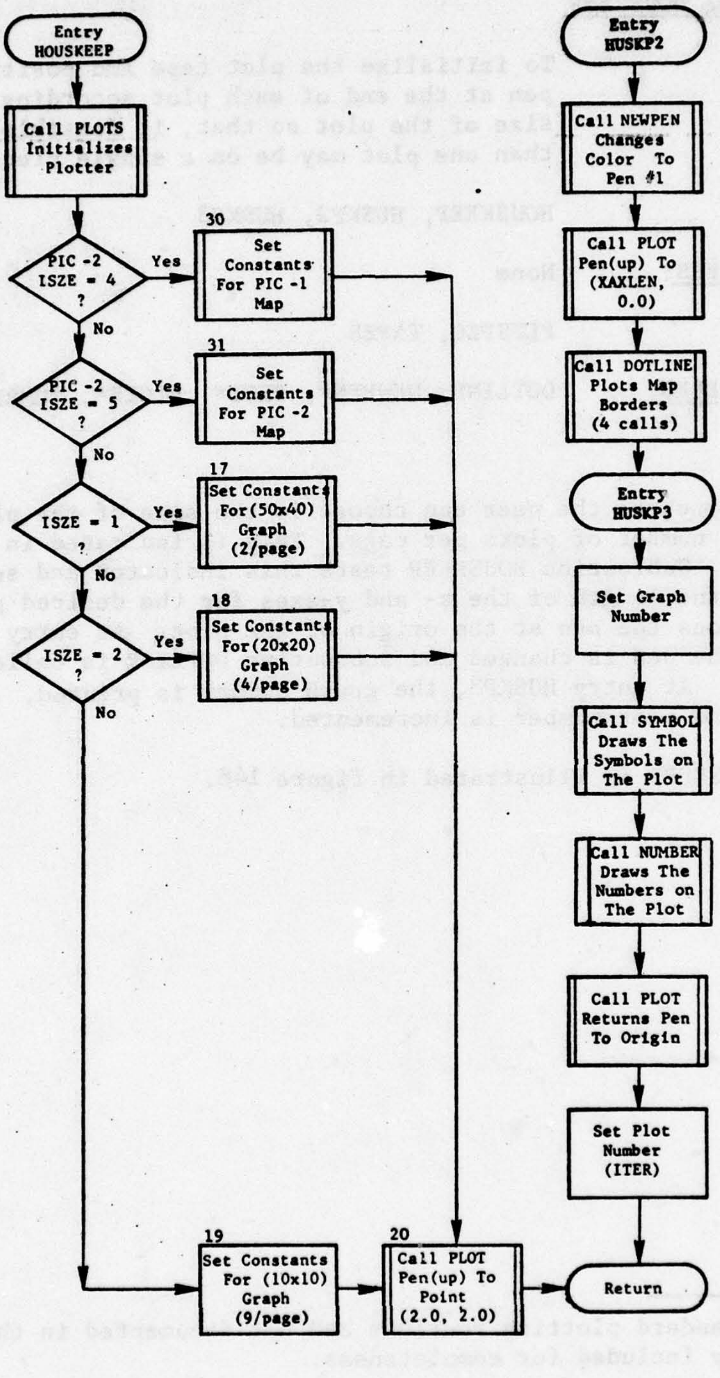


Figure 148. Subroutine HOUSKEEP

9.20 Function IGET

PURPOSE: To unpack a data item from a storage area according to a specified format.

ENTRY POINTS: IGET

FORMAL PARAMETERS:

- KEY - A key word generated by the function KEYMAKE
- INDEX - An index to the array IARR; the value of IGET will be extracted from IARR(INDEX)
- IARR - The array from which the data will be extracted

COMMON BLOCKS: DATPK

SUBROUTINES CALLED: ABORT

Method:

The variable KEY is compared with ISVKEY (the value of KEY on the previous call to IGET). If they are not equal, then the KEY is unpacked and the variables ITYPE, NBITS, and NSHIFT are set (see description of function KEYMAKE). If KEY and ISVKEY are equal, then it is assumed that ITYPE, NBITS, and NSHIFT have been set by a previous call on IGET.

If the data are unpacked (i.e., ITYPE=4), then IGET is set equal to IARR(INDEX) and the routine returns. If ITYPE≠4, then IARR(INDEX) is shifted right NSHIFT bits, and now the rightmost NBITS are placed in IOUT.

If ITYPE=2, then IGET is set equal to IOUT, and the routine returns. If ITYPE=1, then the proper sign is attached to the value IOUT before setting the value of IGET to IOUT and returning control to the calling program.

If ITYPE does not equal 1, 2, or 4, then the subroutine ABORT is called.

Function IGET is illustrated in figure 149.

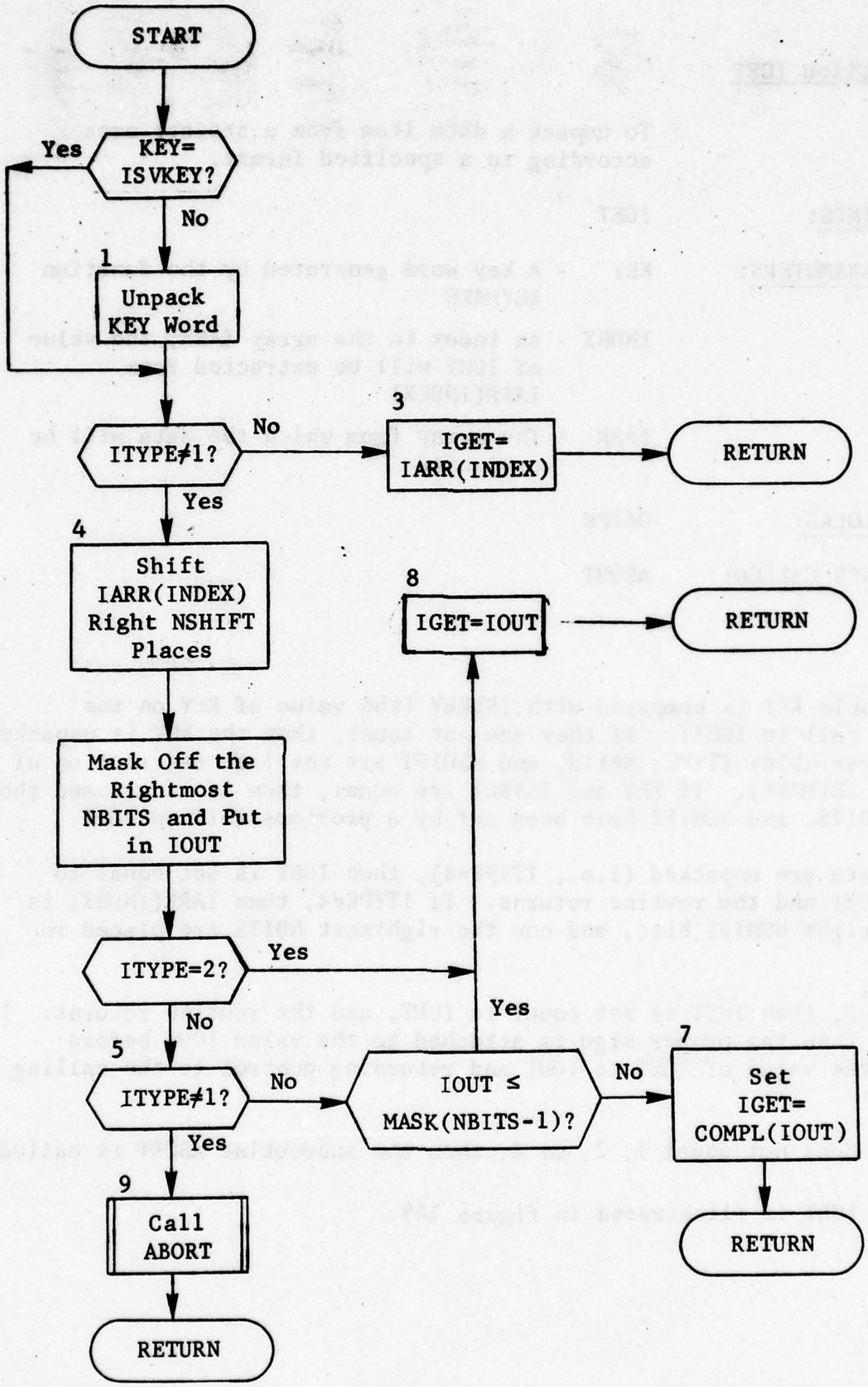


Figure 149. Function IGET

9.21 Function IGETHOB

PURPOSE:	To calculate actual weapon height of burst.
ENTRY POINT:	IGETHOB
FORMAL PARAMETERS:	IHOB - desired height of burst ICNCD - vulnerability number IYLD - weapon yield IN - nonzero if called from PLANSET
COMMON BLOCKS:	None
SUBROUTINE CALLED:	KNOBLANK
CALLED BY:	CALCOMP, INTRFACE

Method:

This function returns the actual weapon height of burst (in hundreds of feet) in R format if called from program PLANOUT. That is, the HOB is contained in the last three characters of the returned value in BCD code. The scaled height of burst is returned if called from program PLANSET. The formal parameter IHOB is an integer variable set to zero for ground burst and one for optimal air burst. The vulnerability number is contained as the last four BCD characters of formal parameter ICNCD if called from PLANOUT. (The country location code CNTRYLOC is the first two characters of this parameter.) The vulnerability is contained in the first four BCD characters of ICNCD if called from PLANSET. The weapon yield in kilotons is contained in formal parameter IYLD in R5 format (right-justified with zero fill to left).

The method of computing actual HOB is described in the Analytical Manual, Volume II, Chapter 2, Calculation of Actual Height of Burst. Function IGETHOB merely implements that calculation. The code for calculating the adjusted vulnerability number is very similar to that used in subroutine VLRADP of program PLANSET. Function KNOBLANK is used to convert the actual height of burst from integer hundreds of feet to BCD code in R format.

The input vulnerability code is decoded into the appropriate vulnerability number VN, the letter ("P" or "Q"), and the K-factor XK. The cube root of the yield in megatons is used to calculate the adjusted vulnerability number AVN. The scaled height burst SHOB is determined by a series of IF statements which use the table specified in the Analytical Manual as referenced in the preceding paragraph. Finally, the actual height of burst (AHOB) is calculated by multiplying by the scaled yield.

Figure 150 illustrates function IGETHOB.

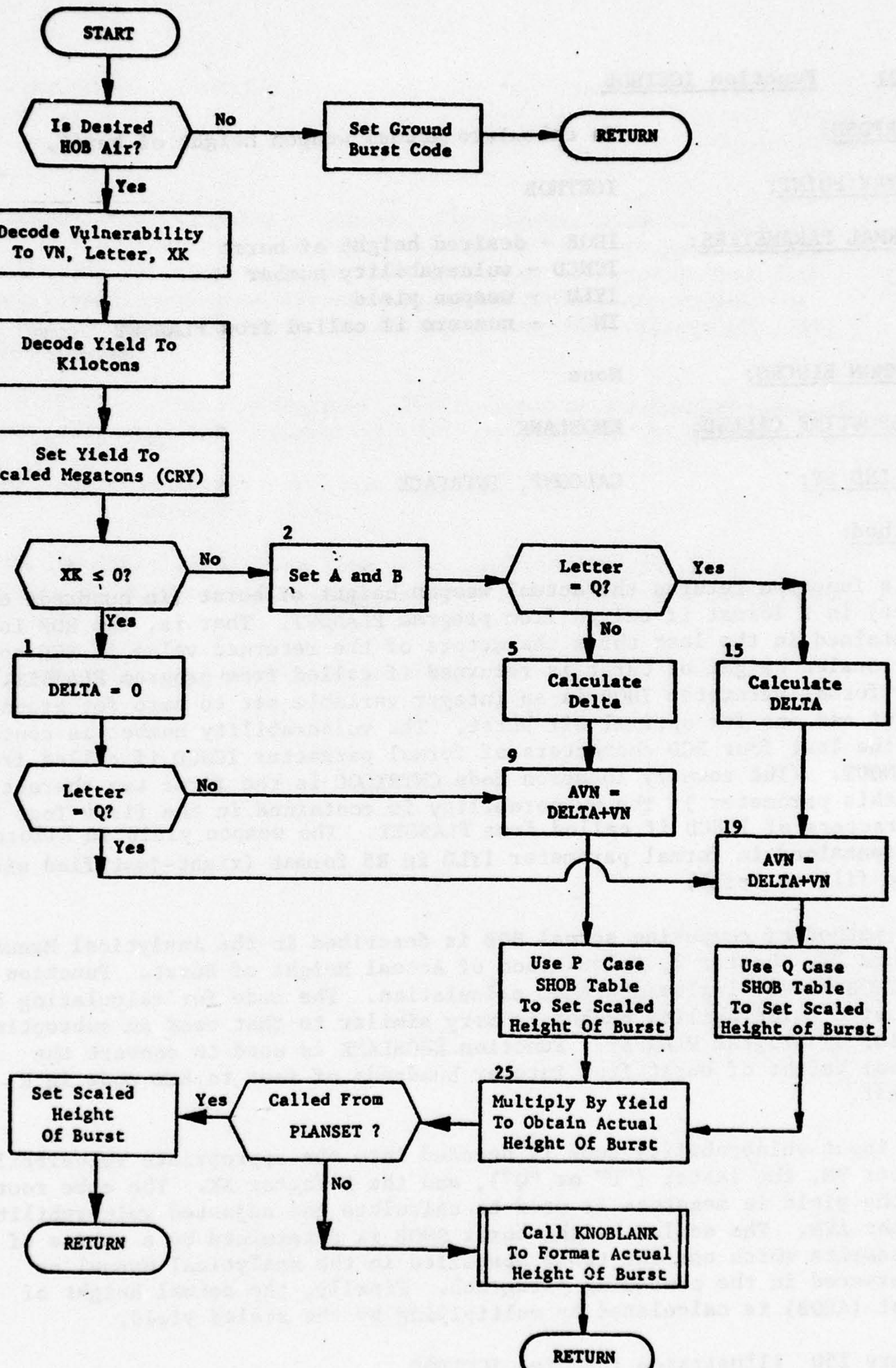


Figure 150. Function IGETHOB

9.22 Subroutine INTERP

PURPOSE:	To find by interpolation the point (SR, TR) located some given fraction of the distance from the point (S1, T1) to the point (S2, T2).
ENTRY POINTS:	INTERP
FORMAL PARAMETERS:	None
COMMON BLOCKS:	POLITE
SUBROUTINES CALLED:	INTRPGC

Method:

The parameters which control the interpolation are contained in common /POLITE/ as follows.

a. Input parameters:

- S1 - latitude of first point
- T1 - longitude of first point
- S2 - latitude of second point
- T2 - longitude of second point
- FACTOR - fraction of distance to be interpolated

b. Output parameters:

- SR - latitude of interpolated point
- TR - longitude of interpolated point

All latitudes and longitudes are carried internally in the QUICK system in the following format:

North latitude	0.	(Equator) to +90. (North Pole)
South latitude	0.	(Equator) to -90. (South Pole)
East longitude	180.	to 360. (Greenwich meridian)
West longitude	0.	(Greenwich meridian) to 180.

The variable FACTOR determines the fraction of the distance from the first point to the second that is equal to the distance from the first point to the interpolated point.

Subroutine INTERP first determines whether the fraction FACTOR is within the interval $0 < \text{FACTOR} < 1$. If not, the result (SR, TR) is set to (S1, T1) when FACTOR ≤ 0, or to (S2, T2) when FACTOR ≥ 1, and the subroutine returns.

When FACTOR is within range, however, the interpolation is to be performed. First, the difference in longitude of the two input points, T12 = T2 - T1, is computed. If that difference is greater than 2.8 degrees, INTERP calls the utility subroutine INTRPGC to perform the interpolation along the great circle route from (S1, T1) to (S2, T2), and then returns.

If $|T12|$ is less than 2.8 degrees, INTERP performs a straight-line, or Mercator, interpolation between (S1, T1) and (S2, T2) by putting:

SR = S1 + FACTOR(S2 - S1) and
TR = T1 + FACTOR(T2 - T1).

When the resultant TR is less than zero, 360 degrees are added to it; similarly, 360 degrees are subtracted from a TR which is greater than 360 degrees.

Subroutine INTERP is illustrated in figure 151.

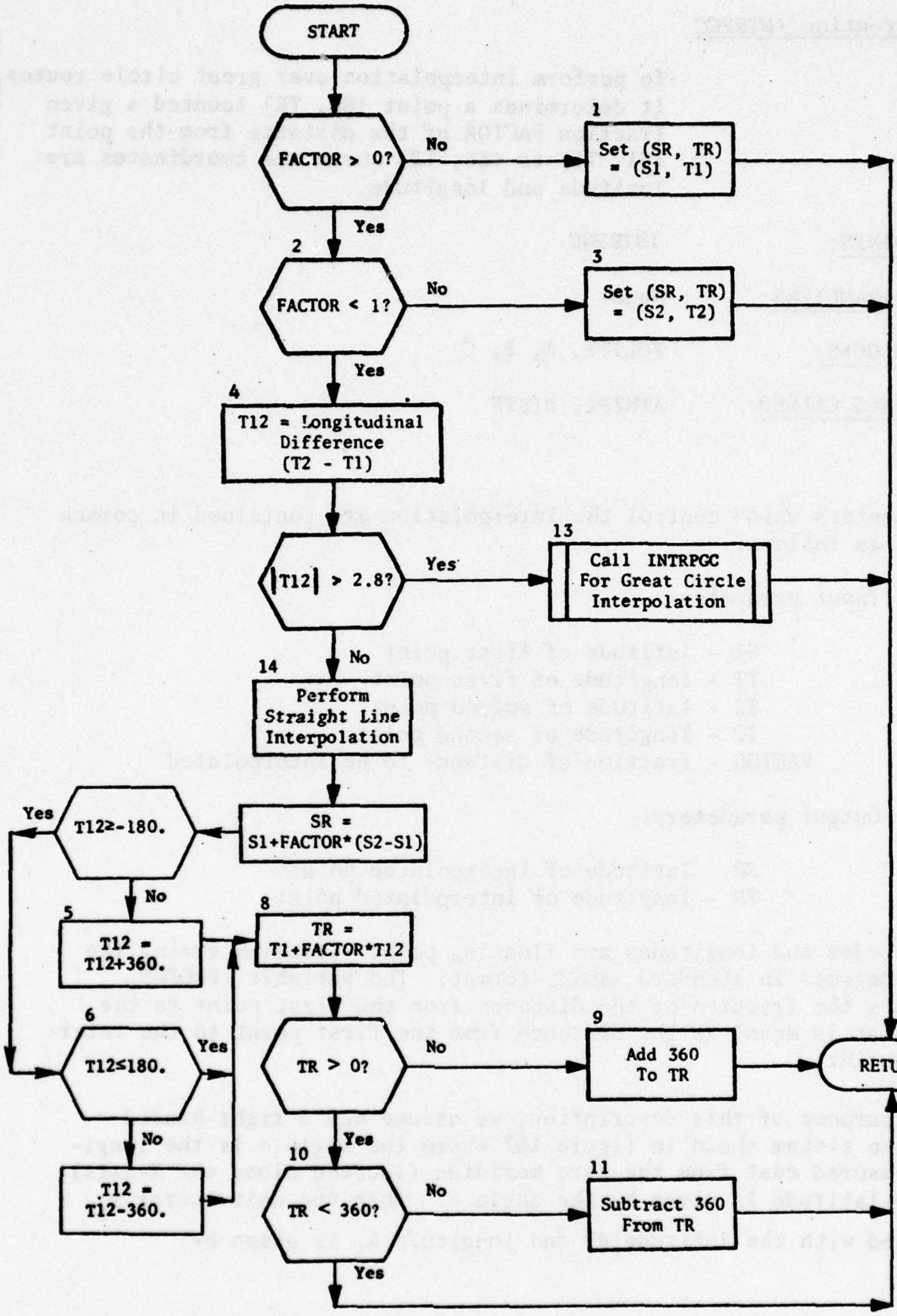


Figure 151. Subroutine INTERP

9.23 Subroutine INTRPGC

PURPOSE: To perform interpolation over great circle routes. It determines a point (SR, TR) located a given fraction FACTOR of the distance from the point (S1, T1) to (S2, T2) where the coordinates are latitude and longitude.

ENTRY POINTS: INTRPGC

FORMAL PARAMETERS: None

COMMON BLOCKS: POLITE, A, B, C

SUBROUTINES CALLED: ATN2PI, DISTF

Method:

The parameters which control the interpolation are contained in common /POLITE/ as follows:

a. Input parameters:

S1 - latitude of first point
T1 - longitude of first point
S2 - latitude of second point
T2 - longitude of second point
FACTOR - fraction of distance to be interpolated

b. Output parameters:

SR - latitude of interpolated point
TR - longitude of interpolated point

All latitudes and longitudes are floating point variables giving the data in degrees in standard QUICK format. The variable, FACTOR, determines the fraction of the distance from the first point to the second that is equal to the distance from the first point to the interpolated point.

For the purpose of this description, we assume now a right-handed coordinate system shown in figure 152 where the angle α is the longitude, measured east from the zero meridian (located along the X-axis) Here the latitude is given by the angle δ. Then the unit vector r_i associated with the latitude δ_i and longitude α_i is given by:

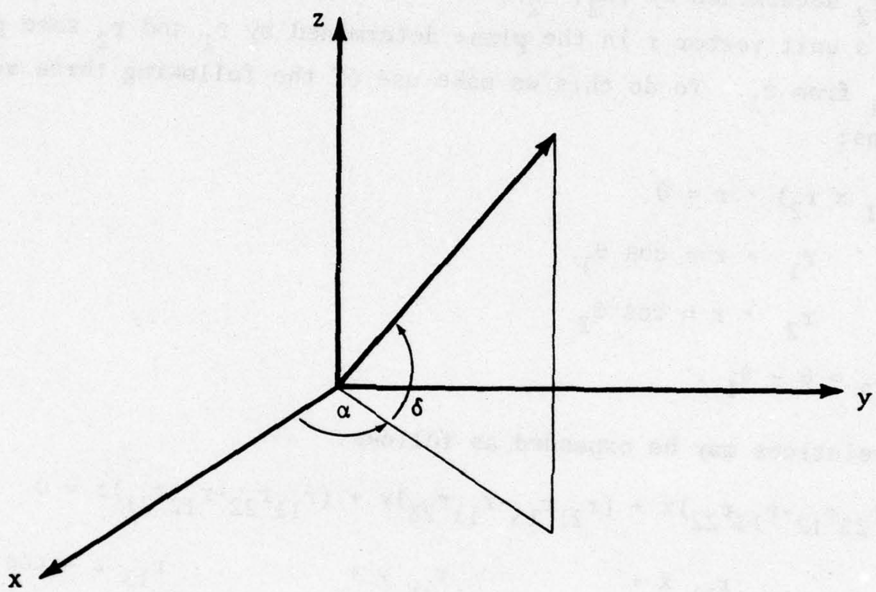


Figure 152. Coordinate System for INTRPGC

$$r_i = \begin{pmatrix} \cos\alpha_i \cos\delta_i \\ \sin\alpha_i \cos\delta_i \\ \sin\delta_i \end{pmatrix}$$

Assume we are given the unit vector r_1 located at (α_1, δ_1) and the unit vector r_2 determined by (α_2, δ_2), and the angle θ between them. We wish to find a unit vector r in the plane determined by r_1 and r_2 some given angle θ_1 from r_1. To do this we make use of the following three vector relations:

$$(r_1 \times r_2) \cdot r = 0$$
$$r_1 \cdot r = \cos\theta_1$$
$$r_2 \cdot r = \cos\theta_2$$

where $\theta_2 = \theta - \theta_1$.

These relations may be expanded as follows:

$$(r_{23}r_{12}-r_{13}r_{22})x + (r_{21}r_{13}-r_{11}r_{23})y + (r_{11}r_{22}-r_{12}r_{21})z = 0$$
$$r_{11}\,x + r_{12}\,y + r_{13}\,z = \cos\theta_1$$
$$r_{21}\,x + r_{22}\,y + r_{23}\,z = \cos\theta_2$$

where r = (x,y,z). This equation has a unique solution in x, y, and z provided $\theta \neq 0$ or $\theta \neq \pi$, since the determinant:

$$(r_1 \times r_2) = \sin^2\theta$$

is not equal to 0. Now since the resultant vector r is:

$$r = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \cos\alpha\cos\delta \\ \sin\alpha\cos\delta \\ \sin\delta \end{pmatrix}$$

then the desired point (α, δ) is given by:

$$\delta = \sin^{-1} Z$$

$$\alpha = \tan^{-1}\left(\frac{y}{x}\right)$$

where the value of the arc tangent is not necessarily its principal value. (Function ATN2PI is used to calculate this value.)

To obtain θ and θ_1 when FACTOR is known, observe that:

if: D is the great circle distance from the point (α_1, δ_1) to the point (α_2, δ_2) and

R is the radius of the earth

then: $\theta = D/R$

$\theta_1 = \text{FACTOR}\cdot\theta$

Subroutine INTRPGC is illustrated in figure 153.

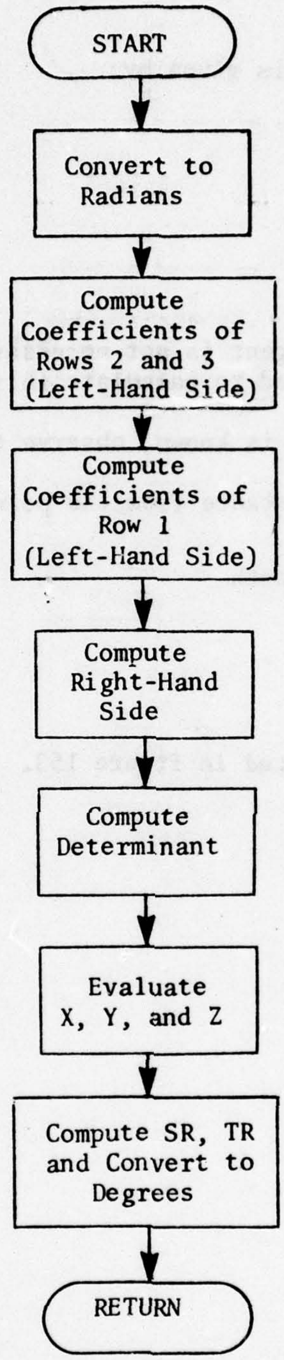


Figure 153. Subroutine INTRPGC

9.24 Subroutine INTPIECE

PURPOSE:	To interpolate between two points where one falls on and the other off the graph.	
ENTRY POINTS:	INTPIECE	
FORMAL PARAMETERS:	XOUT	- x-coordinate of point off graph
	YOUT	- y-coordinate of point off graph
	XIN	- x-coordinate of point on graph
	YIN	- y-coordinate of point on graph
	XBNDRY	- Indicator for XOUT (= -1 if XOUT is within limits of x-axis; = 0 if XOUT off zero edge; = length of x-axis if XOUT off right hand border)
	YBNDRY	- Indicator for YOUT (= -1 if YOUT is within limits of y-axis; = 0 if YOUT off zero edge; = length of y-axis if YOUT off top border
	XEDGE	- Computed x-coordinate of interpolated point
	YEDGE	- Computed y-coordinate of interpolated point
COMMON BLOCKS:	None	
SUBROUTINES CALLED:	None	

Method:

The slope of the connecting line between the two points is computed. If only one of the coordinates of the point off the graph is outside its limits, the point where this line crosses the border is computed and adjusted by the slope. If both coordinates of the point off the graph are outside their limits, the slope of the line connecting the corner of the map and the point on the graph has to be computed and compared with the slope of the line between the two points to determine on which border this line crosses it. Then the border point can be interpolated as above.

Subroutine INTPIECE is illustrated in figure 154.

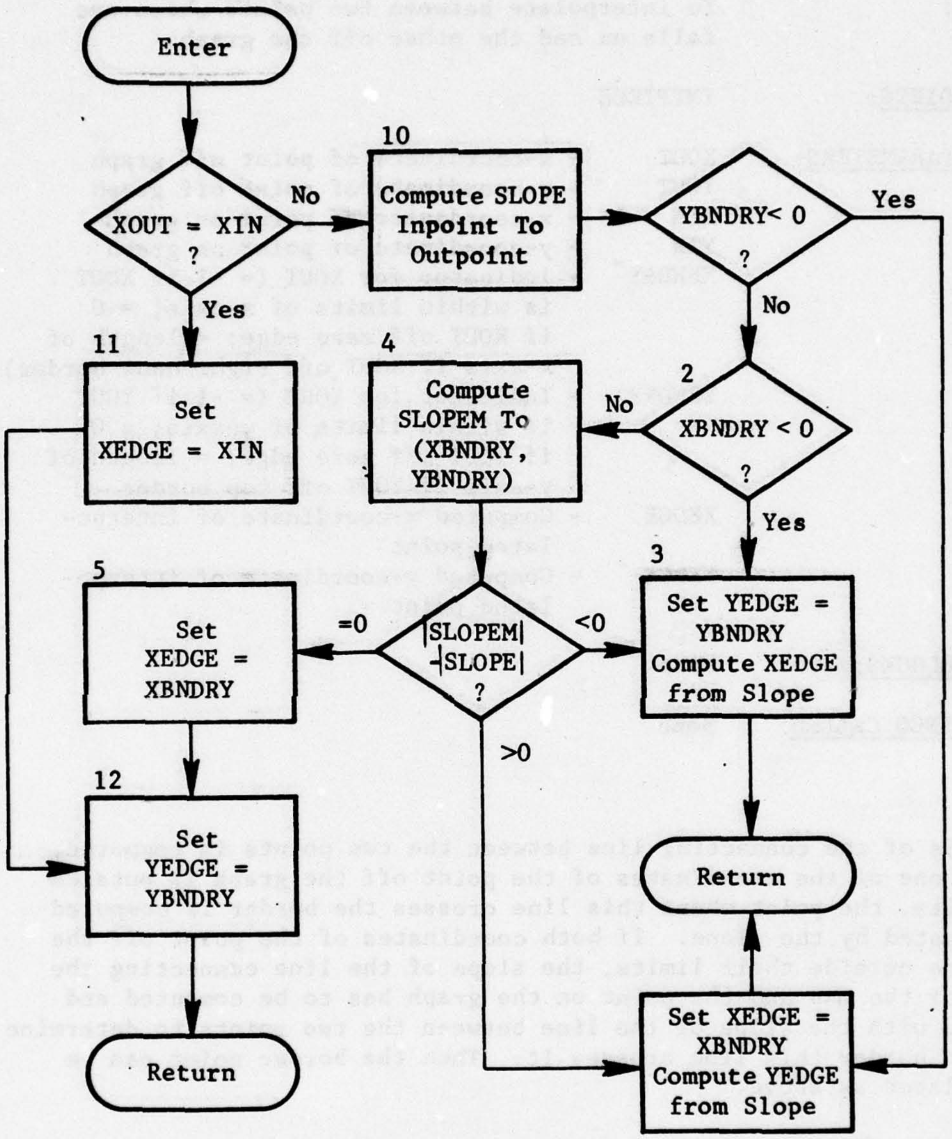


Figure 154. Subroutine INTPIECE

9.25 Subroutine IORFL

PURPOSE:	To check input value and convert it to floating point if it is integer
ENTRY POINTS:	IORFL
FORMAL PARAMETERS:	X: Value to be checked and, if necessary, converted
COMMON BLOCKS:	None
SUBROUTINES CALLED:	None

Method:

First the input value is checked to see if it is a floating point zero. If so the routine exits. If not, bits 8 and 9 of the value are compared. If they are equal the value is integer and is converted to floating point.

Subroutine IORFL is illustrated in figure 155.

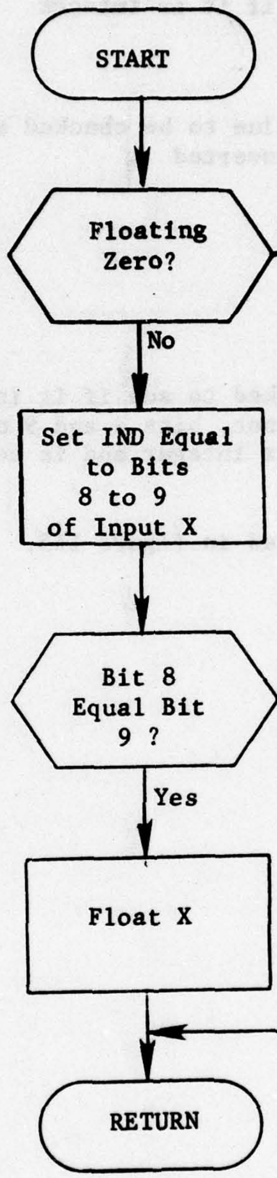


Figure 155. Subroutine IORFL

9.26 Subroutine IPUT

PURPOSE:	To pack a given data item into a storage area according to a specified format.
ENTRY POINTS:	IPUT
FORMAL PARAMETERS:	KEY - A key word generated by the function KEYMAKE
	INDEX - An index to the array IARR; IVAL will be packed into the word IARR(INDEX)
	IVAL - The data word which is to be packed
	IARR - The array into which these data are to be packed
COMMON BLOCKS:	DATPK
SUBROUTINES CALLED:	None

Method:

The variable KEY is compared with ISVKEY (the value of KEY on the previous call to IPUT). If they are not equal, then the KEY is unpacked, and the variables ITYPE, NBITS, and NSHIFT are set (see description of function KEYMAKE). If KEY and ISVKEY are equal, then it is assumed that ITYPE, NBITS, and NSHIFT have been set by a previous call on IPUT.

If the data are not to be packed (i.e., ITYPE=4), then the variables IN and MSK are set equal to IVAL and 0, respectively. If ITYPE≠4, then IN is set equal to the rightmost NBITS of IVAL and shifted left NSHIFT bits. MSK is set equal to MAST(NBITS), shifted left NSHIFT bits, and complemented.

The bits of IARR(INDEX) where the data are to be stored are set equal to zero. The variable IN (which is IVAL shifted to its proper position) is now added to IARR(INDEX).

Subroutine IPUT is illustrated in figure 156.

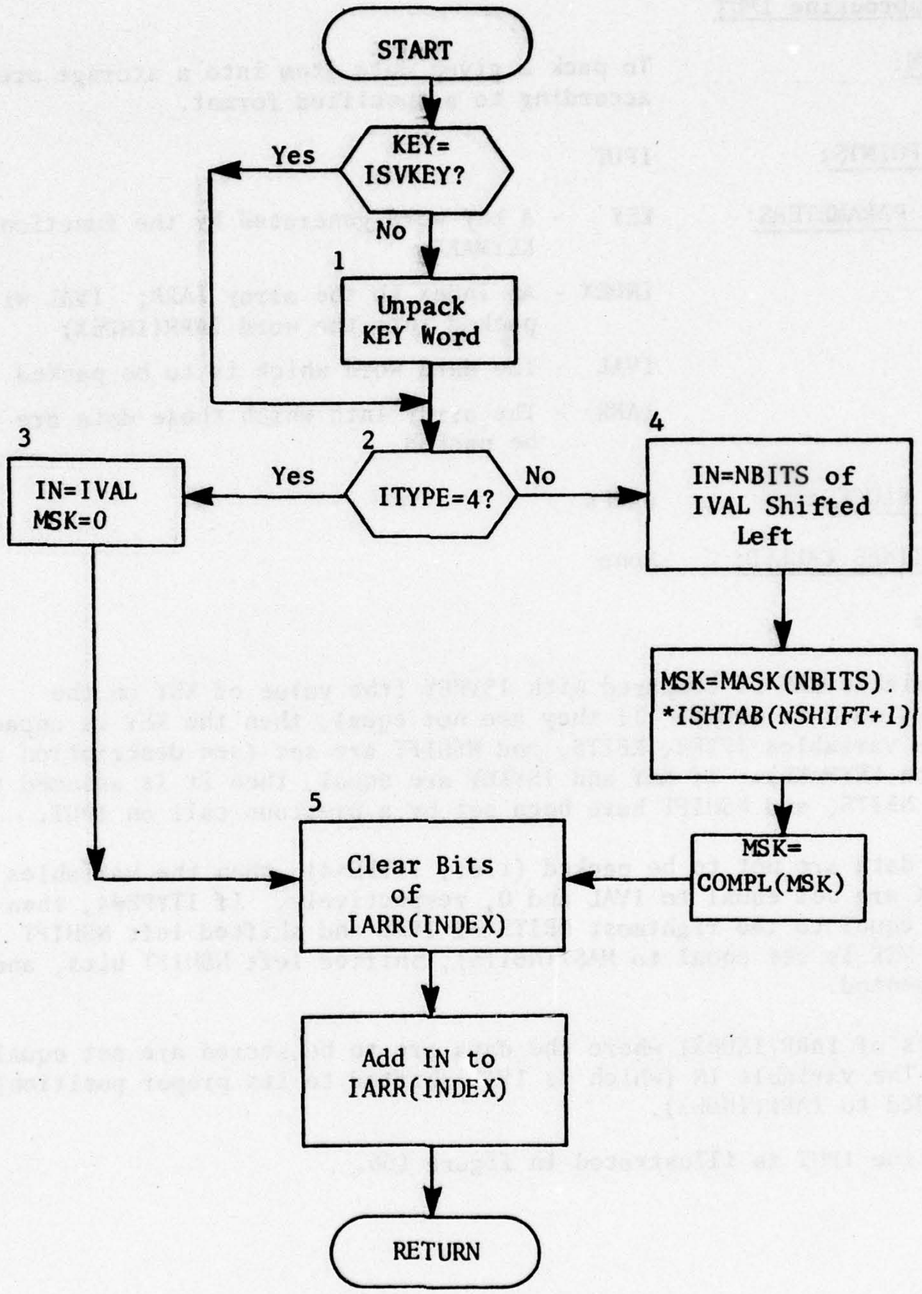


Figure 156. Subroutine IPUT

9.27 Subroutine ISOFF

PURPOSE: To determine if a point is off the plot.

ENTRY POINTS: ISOFF

FORMAL PARAMETERS:

- XN - x-coordinate of point
- YN - y-coordinate of point
- XBNDRY - x value of map boundary outside which point is falling
- YBNDRY - y value of map boundary outside which point is falling
- IYESN - Indicator if point is on the graph (= 0 if point is on the graph; >0 if point is off)

COMMON BLOCKS: PLTSPEC, SNDMIN

SUBROUTINES CALLED: None

Method:

Initially, the parameter IYESN is set to zero; XBNDRY and YBNDRY are set to minus one and left at that value if the point is on the graph. The x- and y-coordinates of the point which is to be tested are compared with the smallest value of x, y (zero) and the largest values of x and y (length of x- and y-axes). If it is found that an x- or y-value is outside the limits, IYESN is incremented by 1; XBNDRY or YBNDRY is set to either zero or the length of the respective axis where the coordinate exceeded its length. Finally, x and y are compared with the minimum values found so far in other calls on ISOFF. If x is smaller than XMIN, it is set to the new x and y is saved in YATXMN. If y is smaller than YMIN, it is set to the new y and x is saved in XATYMN.

Subroutine ISOFF is illustrated in figure 157.

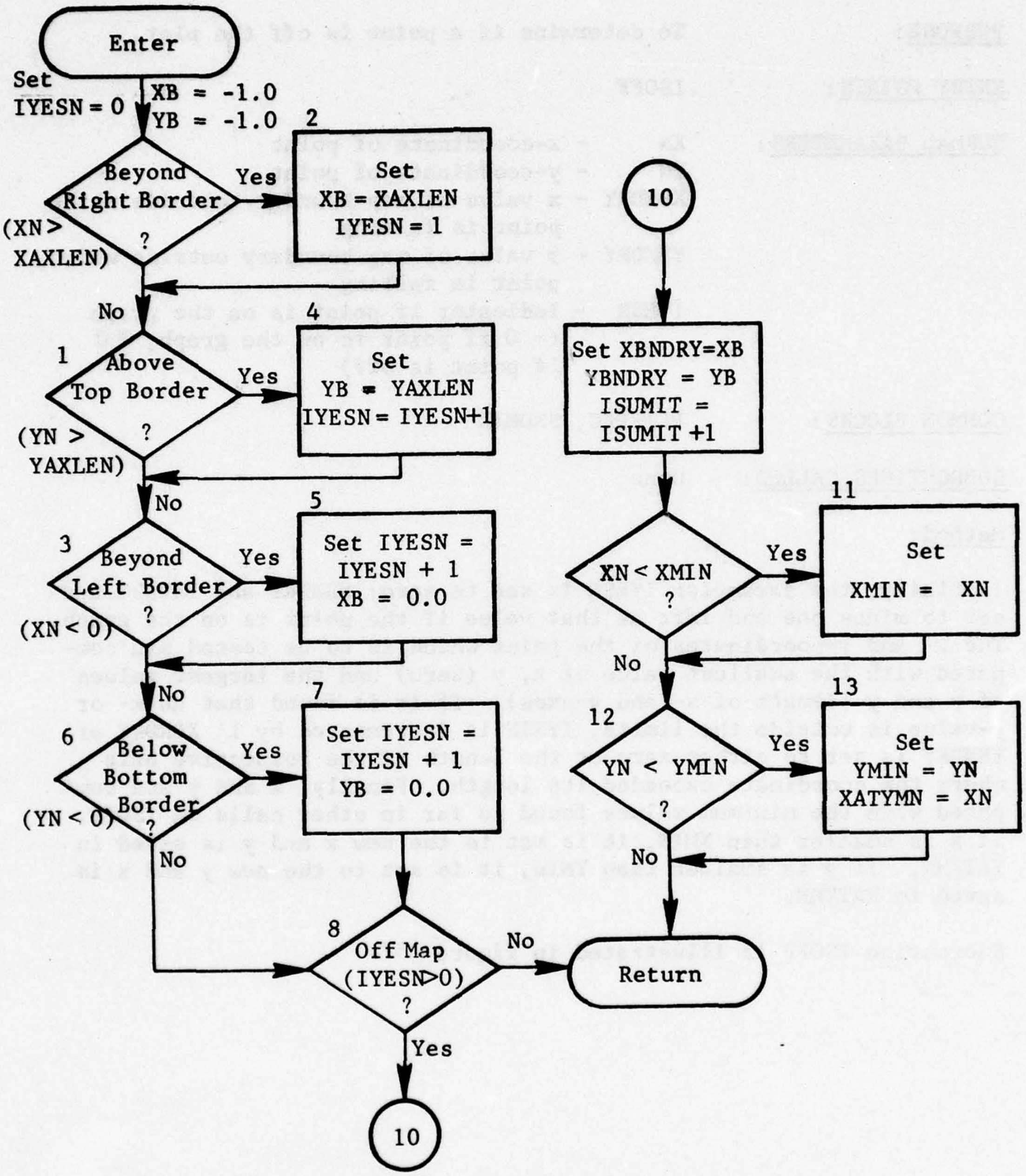


Figure 157. Subroutine ISOFF

9.28 Function ITLE

PURPOSE:	To compare an input value with values in a specified array; to return an index of the first match found, and to return zero if no match is found.
ENTRY POINTS:	ITLE
FORMAL PARAMETERS:	NX - The value for which a match is to be found
	NAR - The array with which NX is to be compared
	NMB - The number of words to be scanned in NAR
COMMON BLOCKS:	None
SUBROUTINES CALLED:	None

Method:

The input search value is compared with each member of the array to be scanned. If no match is found, a value of zero is returned. Otherwise, the index of the element of the array which exactly matches is returned. In the case of more than one match, the smallest index value is returned.

Function ITLE is illustrated in figure 158.

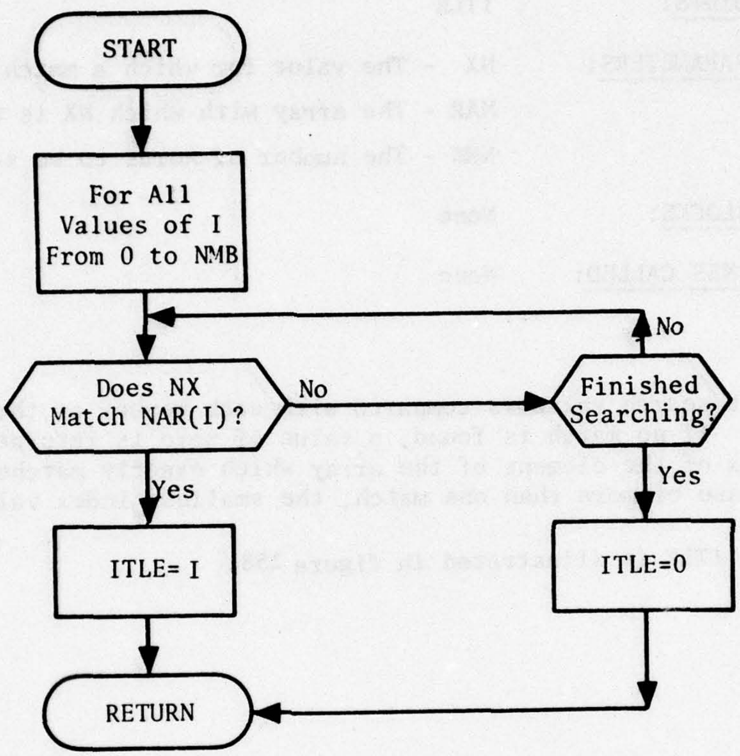


Figure 158. Function ITLE

9.29 Function IWANT

PURPOSE: To determine the position of an item in an array. The search is modified by specifying items not to be searched.

ENTRY POINTS: IWANT

FORMAL PARAMETERS:

- NEED - The item to be matched
- IAR - The array to be searched
- ISTART - The first element in the array to be searched
- IEND - The last element in the array to be searched

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method:

This function searches array IAR from element ISTART to element IEND to find a match for NEED. At each occurrence of an element of IAR with the value = 1R*; i.e., 1-character BCD code for asterisk right-justified with zero fill to the left, the list is not searched for NEED until after the next occurrence of the right-justified asterisk. This search method corresponds to ignoring alphameric value subfields as defined by subroutine GETVALU. The value returned by the function is the index to the element of IAR which is equal to NEED. If no element meets the prescribed conditions, the function returns the value IEND+1

Function IWANT is illustrated in figure 159.

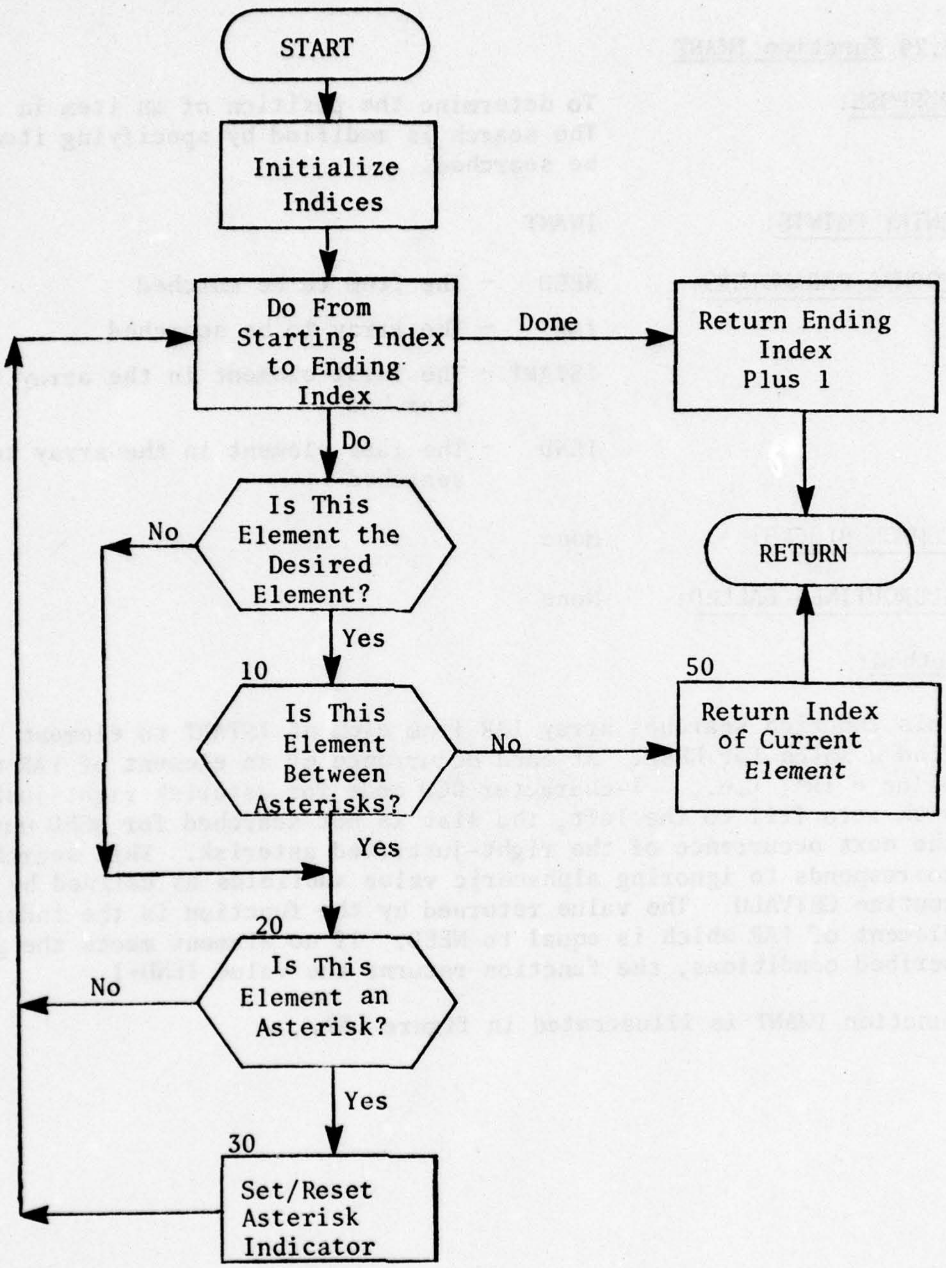


Figure 159. Function IWANT

9.30 Function KEYMAKE

PURPOSE:	To generate a KEY word for input to subroutine IPUT or function IGET.
ENTRY POINTS:	KEYMAKE
FORMAL PARAMETERS:	ITYPE - Specifies the type of packing or unpacking =1 Signed integer =2 Unsigned integer =4 Unpacked word NSHIFT - The number of bits from the rightmost bit of the packed word NBITS - The number of bits in the packed word
COMMON BLOCKS:	None
SUBROUTINES CALLED:	None

Method:

Function KEYMAKE generates a key word which contains the necessary packing and unpacking information for input to IPUT or IGET. The format of the key word is as follows:

Number of Bits	21	6	6	3
	Unused	NSHIFT	NBITS	ITYPE

The variable IWD is set equal to NSHIFT and shifted left six bits. NBITS is added to IWD, and IWD is shifted left three bits. ITYPE is now added to IWD and the result placed in KEYMAKE, which value is returned to the calling program.

Function KEYMAKE is illustrated in figure 160.

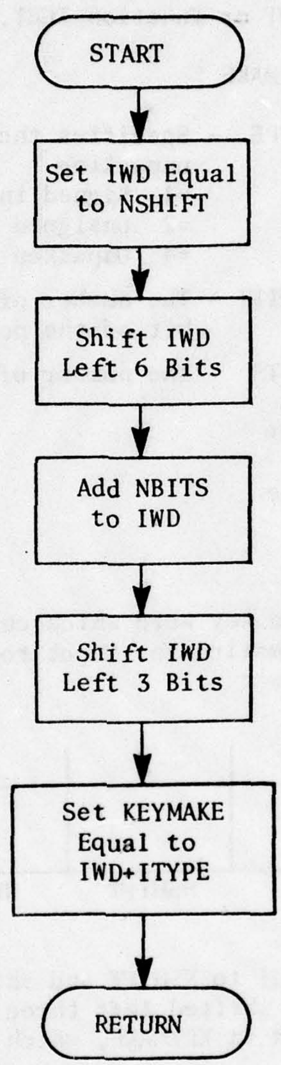


Figure 160. Function KEYMAKE

9.31 Function LATBT

PURPOSE: To convert a floating point latitude or longitude to degrees, minutes, seconds, and direction (CHARACTER*8).

ENTRY POINTS: LATBT, LATCV, LATDT, LATS, LONBT, LONCV, LONDT, LONS

FORMAL PARAMETERS: A - Floating point latitude or longitude

JDIR - Specifies whether latitude or longitude coordinate to be converted

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method:

Function LATBT converts a floating point value (A) of latitude (entry point LATBT) or longitude (entry point LONBT) to the form DDMMSSH or DDDMMSSH, where DD or DDD is degrees, MM is minutes, SS is seconds, and H is the hemisphere code. The function first assigns a hemisphere code to variable LHEM as follows: east longitude (E), west longitude (W), north latitude (N), and south latitude (S). The degrees, minutes and seconds are then computed from A and are packed, along with LHEM, into variable LATBT.

The logic flow of function LATBT is illustrated in figure 161.

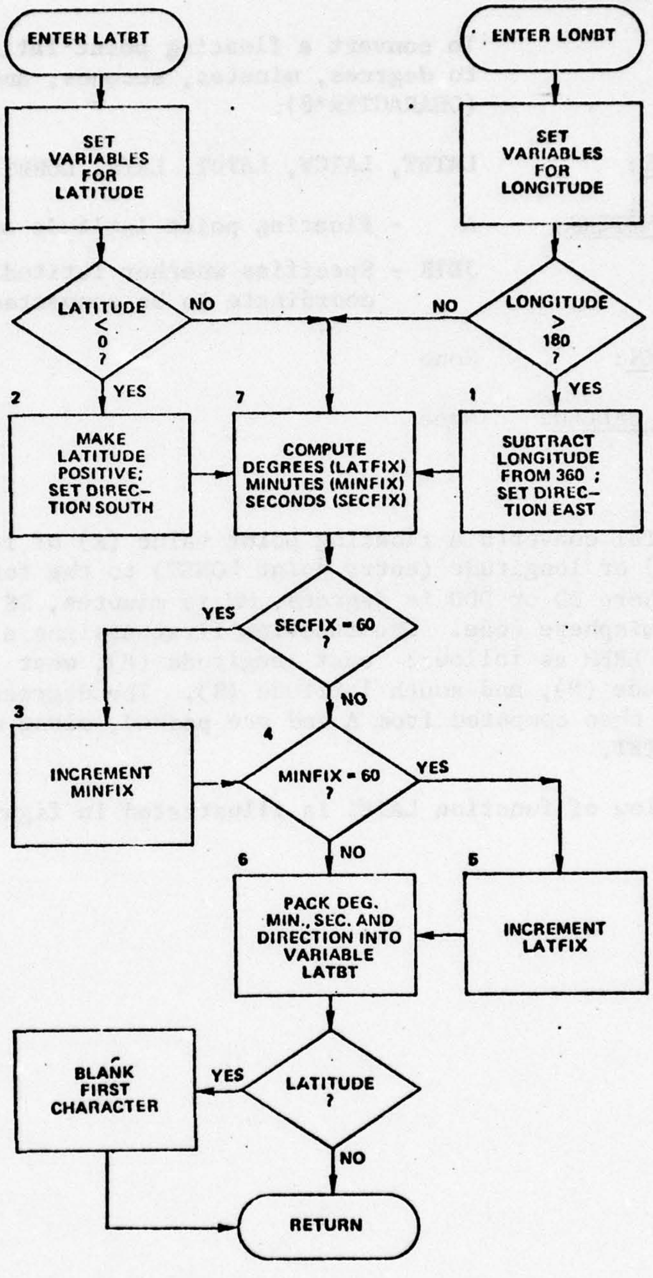


Figure 161. Function LATBT

9.32 Subroutine LINKUP

PURPOSE:	To add to a list of record types any additional types needed to build a retrieval scheme
ENTRY POINTS:	LINKUP
FORMAL PARAMETERS:	LIST: List of record type numbers LISTLN: Length of list KHDR: Record type number of primary header
COMMON BLOCKS:	C10, C20, C30, SCRTCH
SUBROUTINES CALLED:	NEXTTT

Method:

This subroutine is best understood by examining figure 162. Basically, the routine begins with the input LIST and pairs to it the JIST array which contains an identifying number for the 'complex' of record types to which each record type in LIST belongs. A complex (as defined here) is a group of record types which is continuous in that every record type in the group is linked to every other type in the group through one or more data base chains. The objective of the LINKUP process is to add record types to LIST in such a way that all record types end up in the same complex. If an error condition occurs the length of the output LIST is set to zero (LISTLN=0).

In the first part of the process JIST is set to 0 for all but the primary header (KHDR) which is set to 1. Then each element of LIST is examined as follows. If JIST is zero, it is set to the next number in sequence (ICOMP). Then for every chain of which it is master the detail record is compared to LIST. If a match is found and the match entry has JIST=0, JIST is set equal to JIST for the current LIST element. If a match is found and the match entry has JIST=0 then all JIST entries equal either to the match entry for JIST or the current entry for JIST are set equal to the lower of these two.

If no match is found for the detail record type in LIST, it is compared to the auxiliary table KIST. If a match is found in KIST, the KIST entry is removed and added to LIST. Furthermore, all entries equal to the JIST current value or the auxiliary table complex number (MIST) for the KIST entry are set equal to the lower numbered value. This reset is done in both JIST and MIST.

If no match is found for the detail record in either LIST or KIST, the detail is added to KIST and the appropriate value set in MIST. When all entries of LIST have been processed, JIST is checked to see if any entries are not equal to 1. If any are, a more complex process must

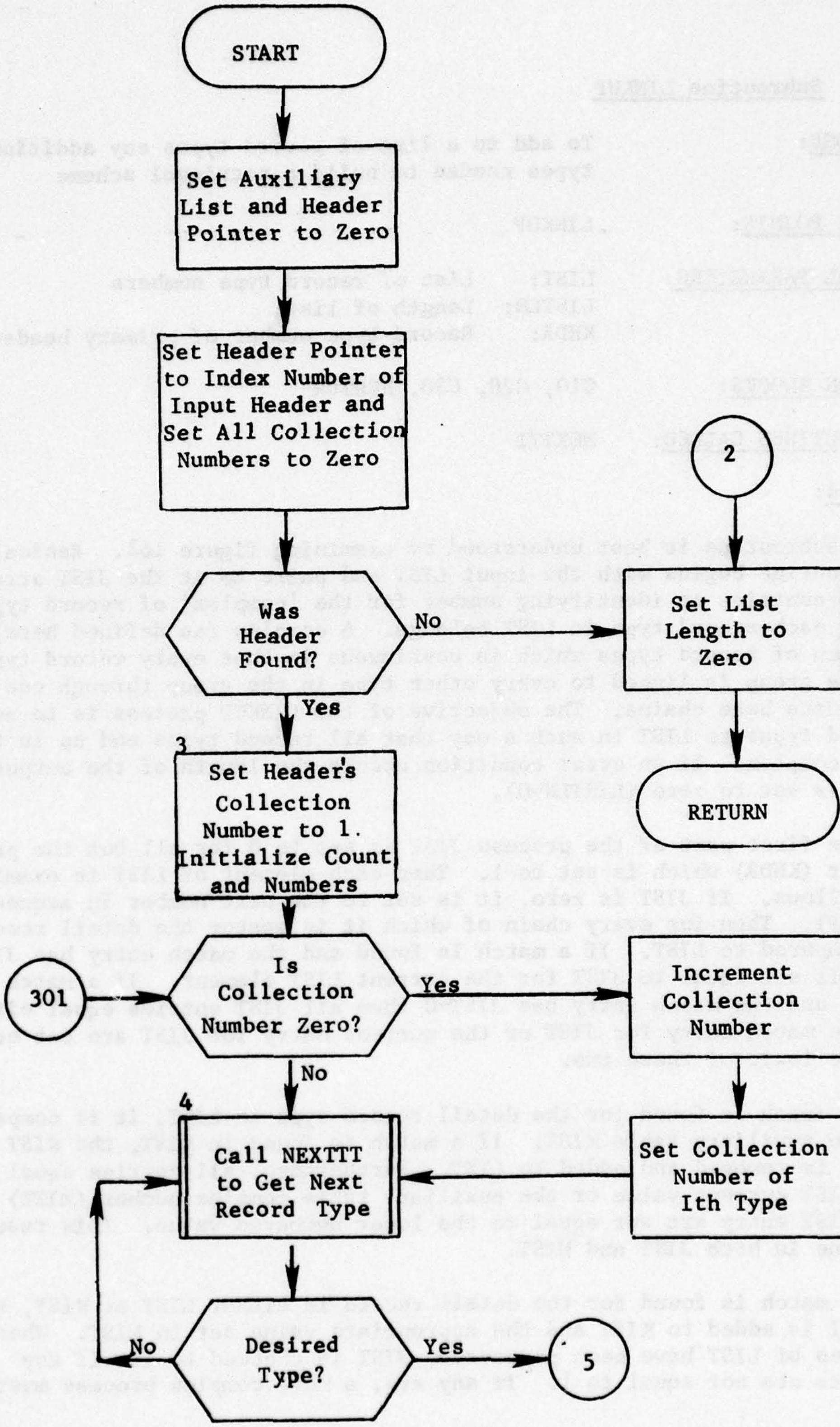


Figure 162. Subroutine LINKUP (Part 1 of 8)

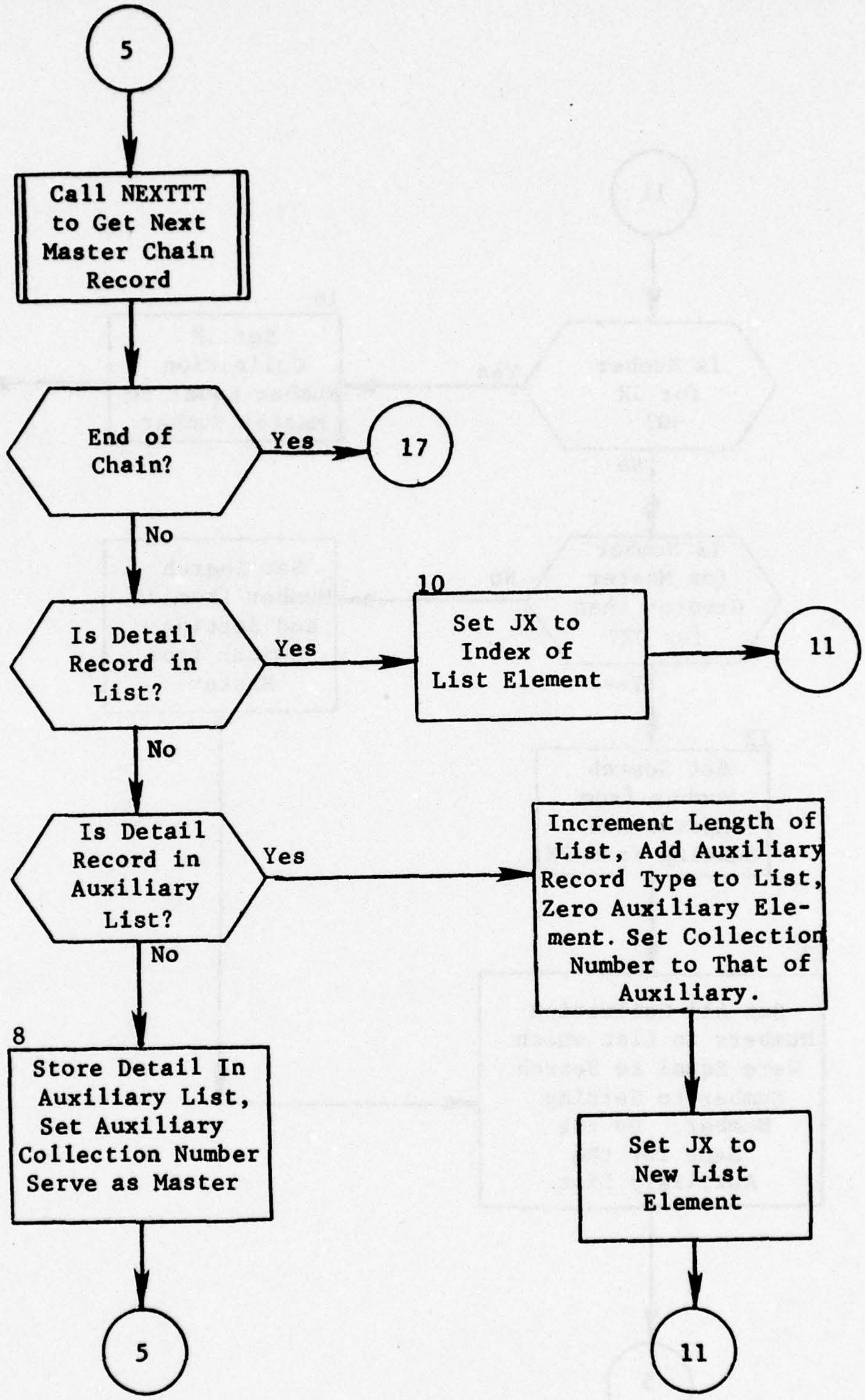


Figure 162. (Part 2 of 8)

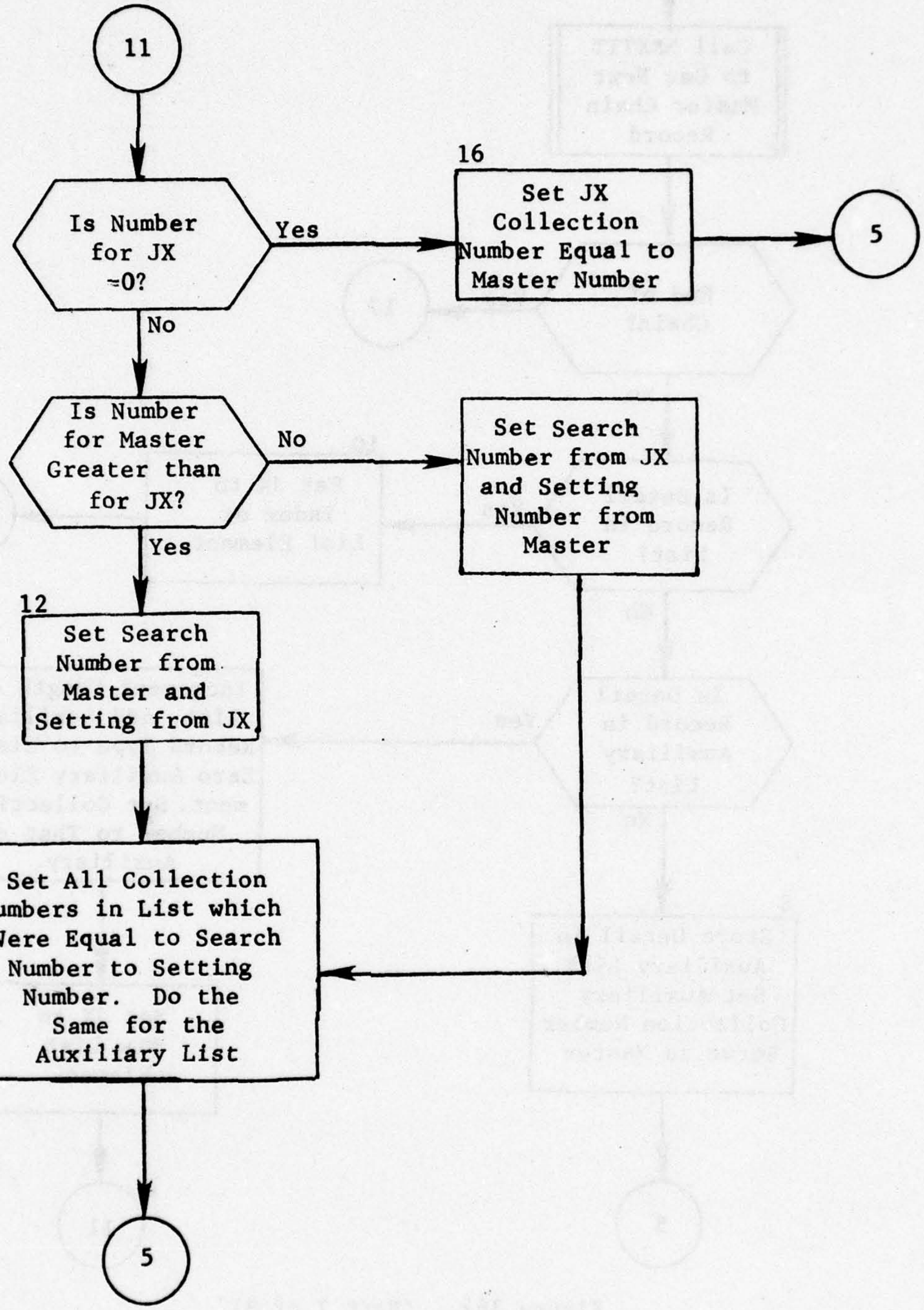


Figure 162. (Part 3 of 8)

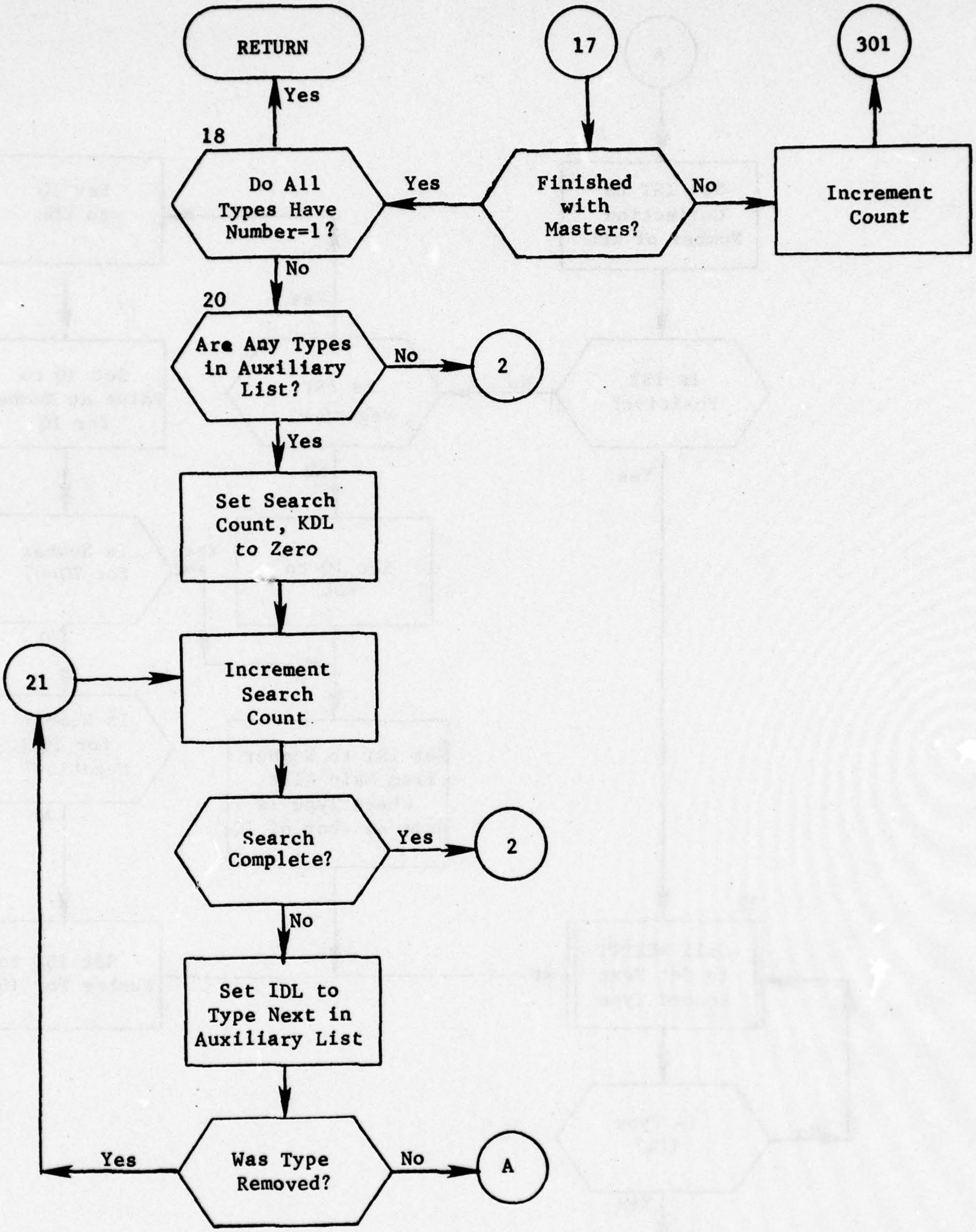


Figure 162. (Part 4 of 8)

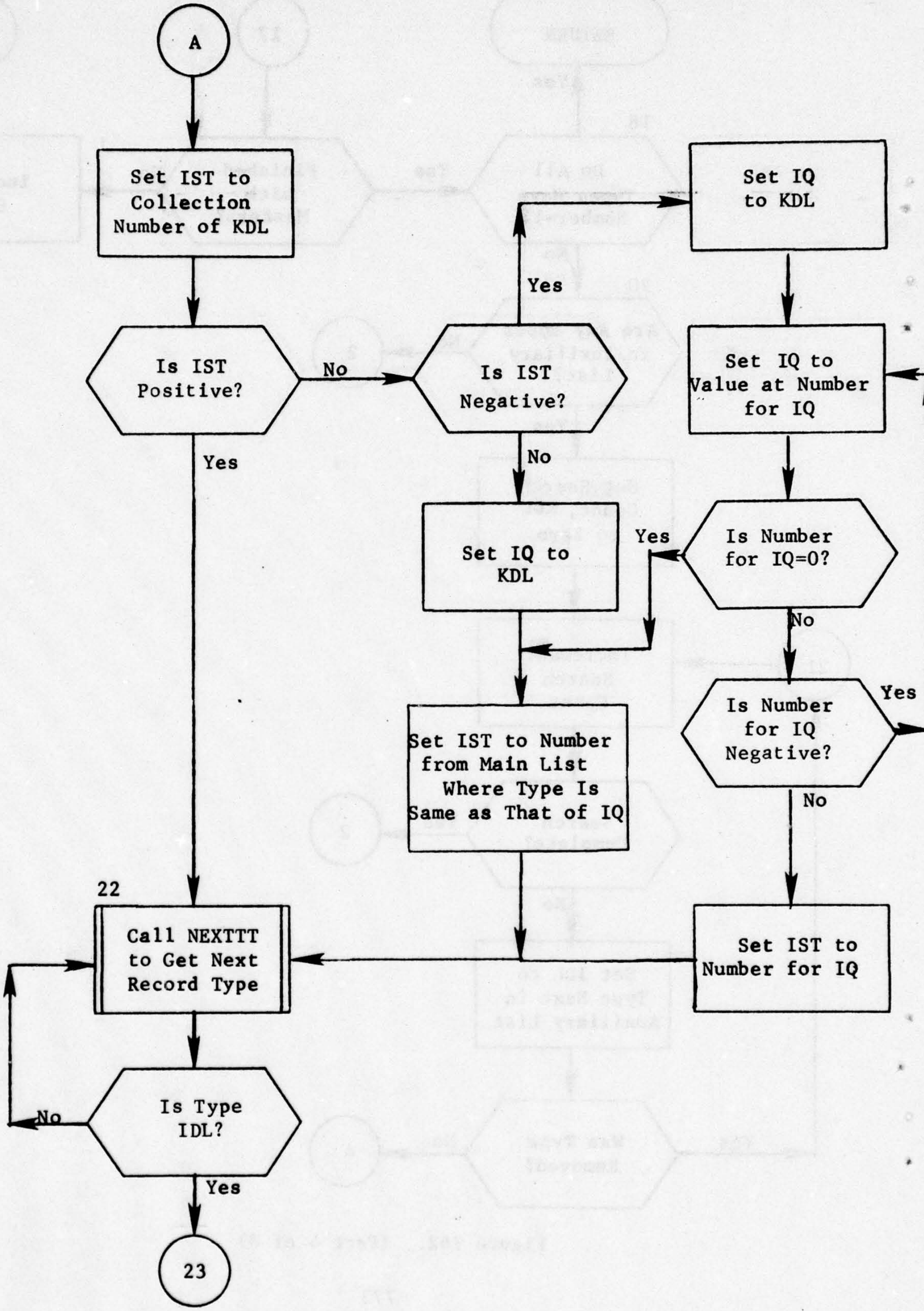


Figure 162. (Part 5 of 8)

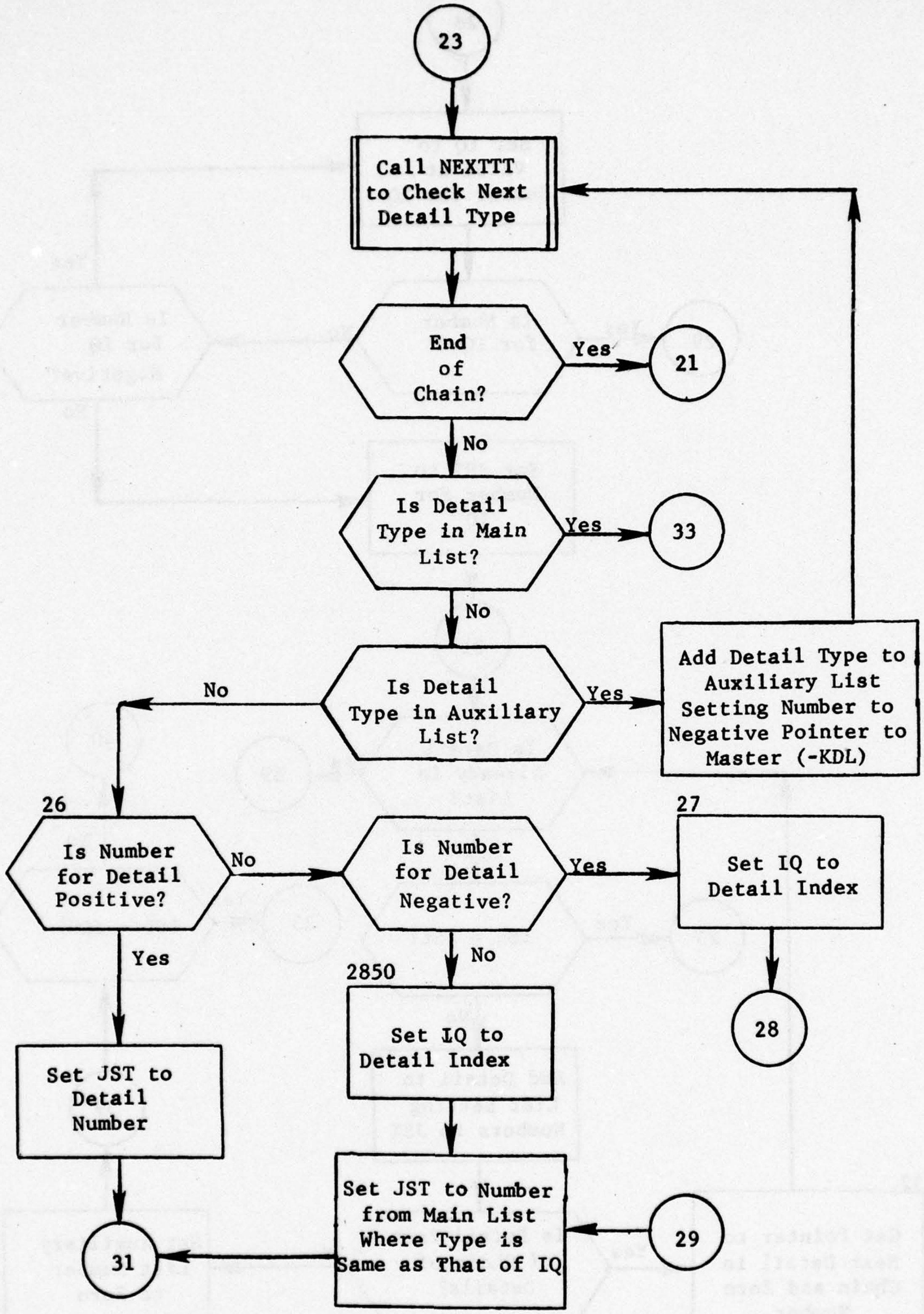


Figure 162. (Part 6 of 8)

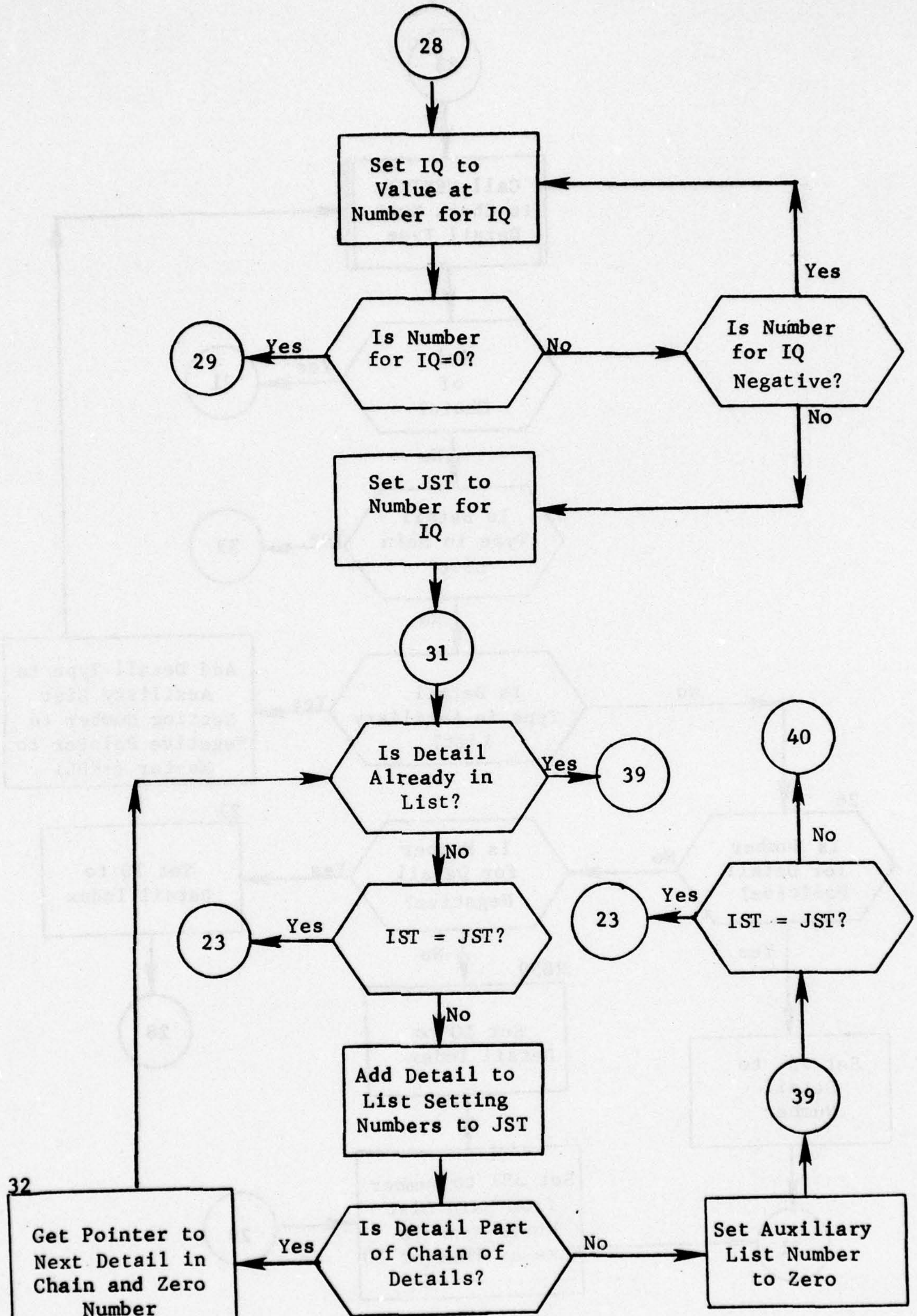


Figure 162. (Part 7 of 8)

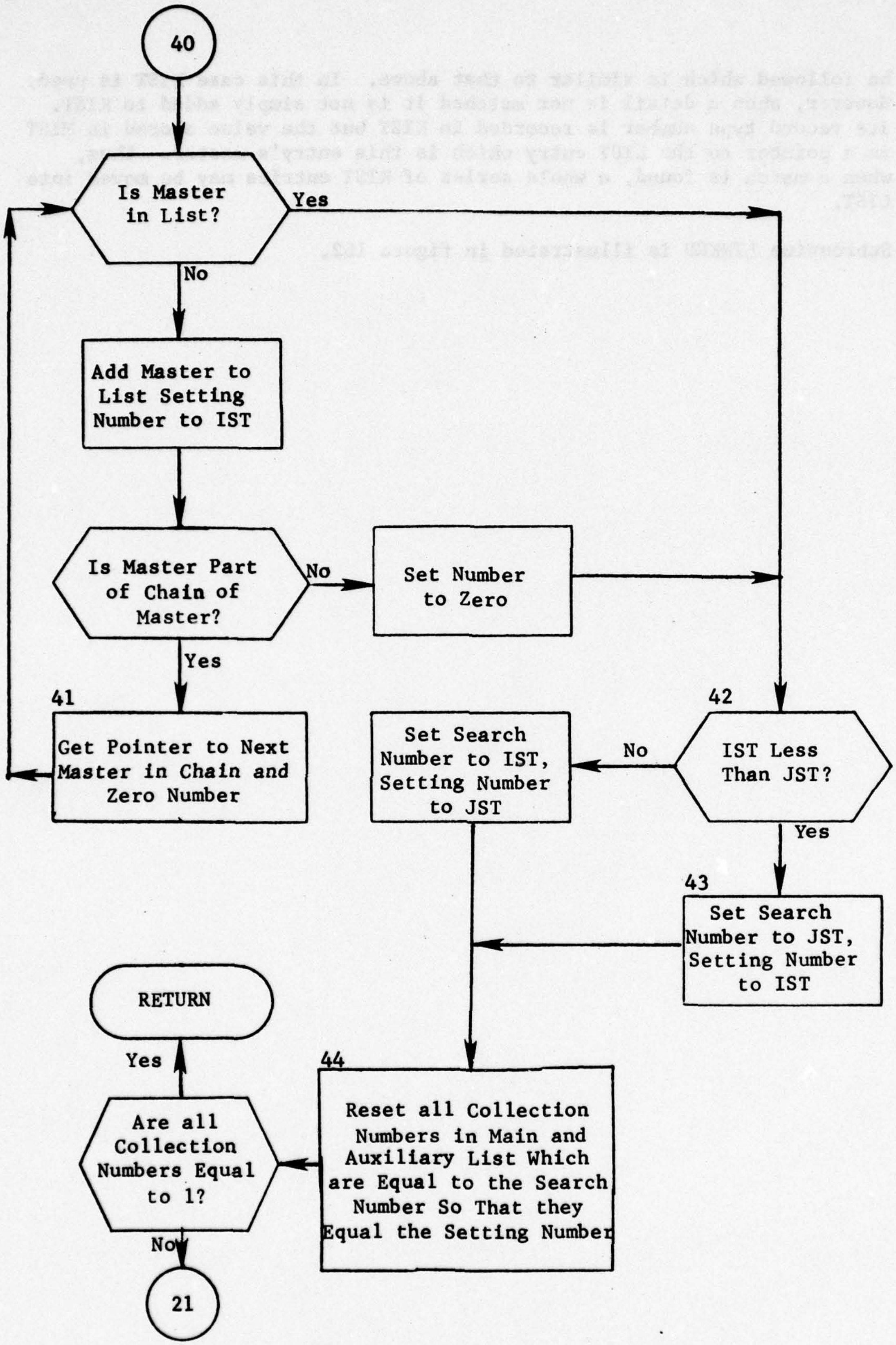


Figure 162. (Part 8 of 8)

be followed which is similar to that above. In this case KIST is used. However, when a detail is not matched it is not simply added to KIST. Its record type number is recorded in KIST but the value stored in MIST is a pointer to the KIST entry which is this entry's master. Thus, when a match is found, a whole series of KIST entries may be moved into LIST.

Subroutine LINKUP is illustrated in figure 162.

9.33 Subroutine LREORDER

PURPOSE:	This routine reorders the elements of a packed logical array.
ENTRY POINTS:	LREORDER
FORMAL PARAMETERS:	ISEQ - A sequence array to control reordering N - Number of elements to be reordered LXLOGAR - A logical array to be reordered
COMMON BLOCKS:	None
SUBROUTINES CALLED:	SLOG, GLOG

Method:

This subroutine uses the same method as the utility subroutine REORDER. This extra routine is required for logical arrays on the HIS 6080 computer system. QUICK system logical arrays are packed 36 elements to one computer word on this system and the word manipulation code of subroutine REORDER would not correctly reorder a packed array.

The ISEQ array is a sequence key array of the type produced by subroutine ORDER. It contains the indices of the array LXLOGAR in the order in which they are placed. That is, ISEQ(1) contains the index of the element in LXLOGAR that is to be placed first; ISEQ(2) contains the index of the element that is to be placed second, and so on. The parameter N determines the number of elements to be reordered. At the end of the subroutine, the elements of LXLOGAR have been reordered.

LREORDER stores one element from LXLOGAR in a temporary location. It then reads from ISEQ the element which should go in that position (which may now be considered empty) and moves it, filling the position and creating a new empty cell. Each new empty cell is filled with its proper contents as soon as the original contents have been removed. When the element in temporary storage is required, LREORDER finds another element which is not already in proper sequence, puts it into temporary storage, and continues as before. This process continues until no elements are out of sequence. The contents of ISEQ are returned to the calling program unchanged so that the sequence key can be used again.

Subroutine LREORDER is illustrated in figure 163.

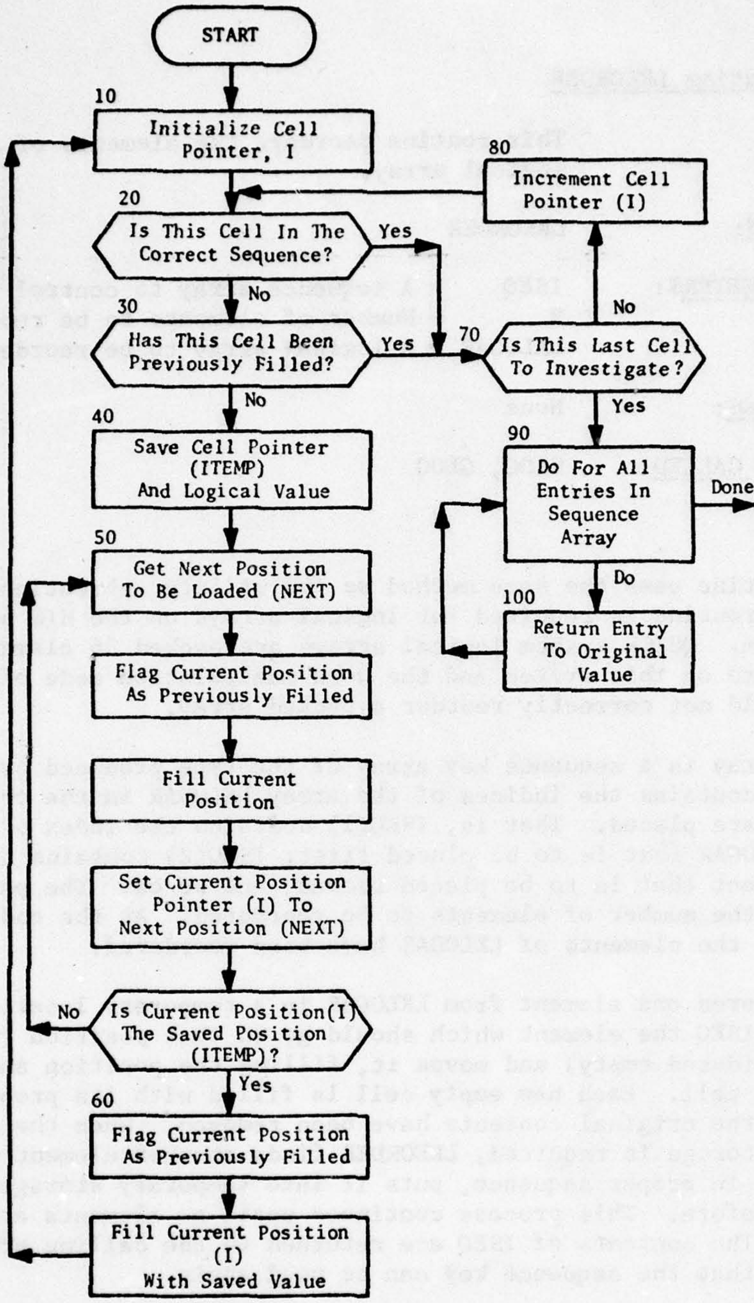


Figure 163. Subroutine LREORDER

9.34 Subroutine MAPEDGE

PURPOSE: To determine if the line being described should go to the edge of a map and begin again on the other side or simply connect the points (only needed for Mercator projection).

ENTRY POINTS: MAPEDGE

FORMAL PARAMETERS: K - Line indicator

COMMON BLOCKS: INMAP, PLTPROJ, XMEDGE

SUBROUTINES CALLED: PIECEIT, PIECENUM

CALLED BY: PLTDATA, SUBPLOT

Method:

The points of the plot which are to be connected may lie on different sides of the map if it is drawn in the Mercator projection. The line indicator, K, is tested if it is desired to connect those points by continuing the line from the other side of the map. If the projection indicator, MERCAT, is not set for Mercator projection, the subroutine exits even if K is set to draw the line as described above. The differences of the values of the x-coordinates and y-coordinates for the points to be connected are computed. Then the line connecting the x-coordinates is tested to determine if it leads off the XMEDGE or zero edge of the map. The variable SIGN is set to 1 to indicate XMEDGE or to -1 to indicate the zero edge. The point along the edge where the line is to continue is computed and subroutines PIECEIT and PIECENUM are called to perform the plotting and labeling.

Subroutine MAPEDGE is illustrated in figure 164.

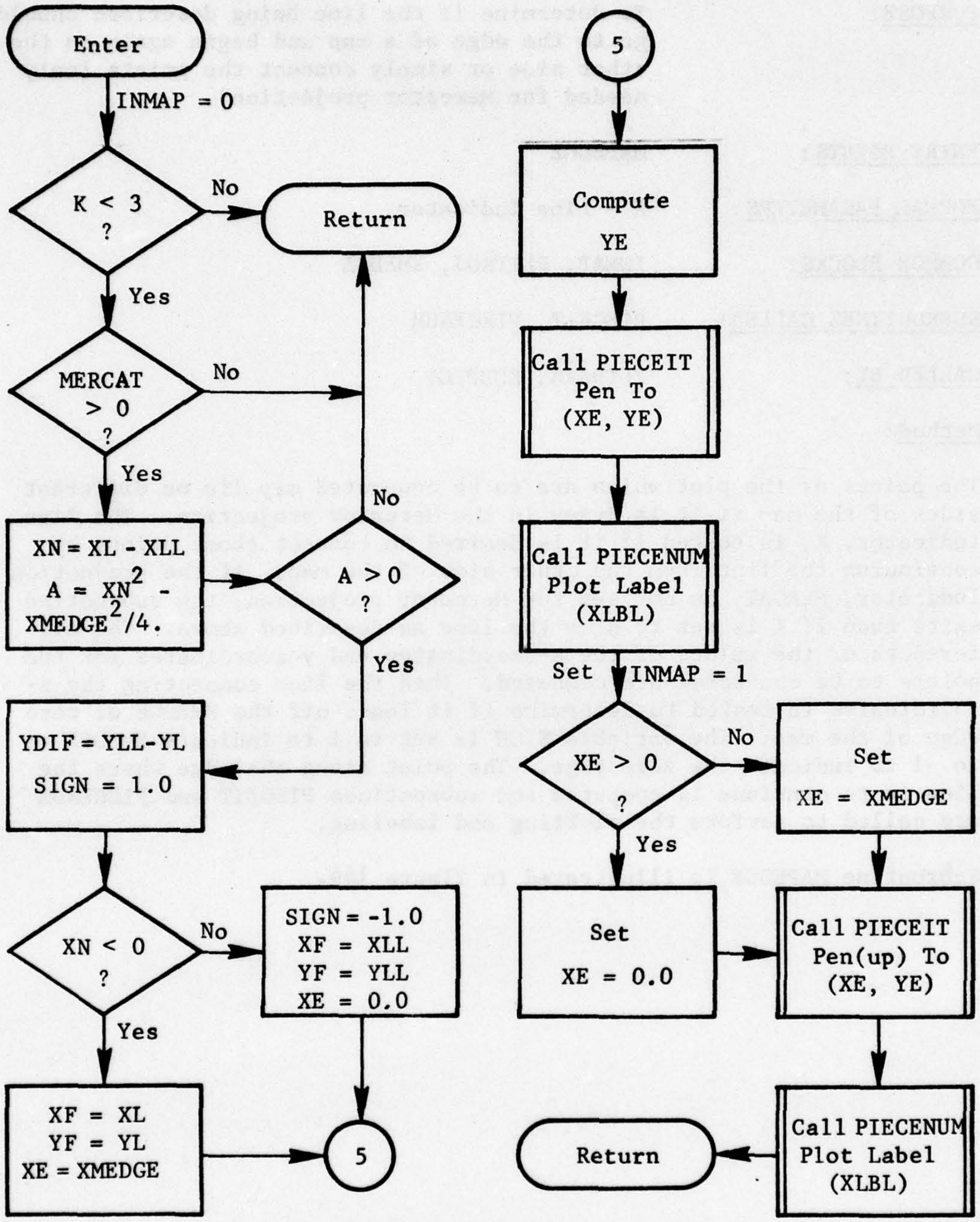


Figure 164. Subroutine MAPEDGE

9.35 Function NUMGET

PURPOSE:	To convert the BCD contents of an array into a signed integer number.
ENTRY POINTS:	NUMGET
FORMAL PARAMETERS:	NIN - The name of the array or variable where the input data are stored
	NCH - The number of characters to be scanned
COMMON BLOCKS:	None
SUBROUTINES CALLED:	None

Method:

Function NUMGET converts integer data, which have been read into an array using alphameric (A6) format for each word, into signed integer numbers with blanks ignored; i.e., suppressed. The number of characters to be scanned and the array or variable name are specified in the calling sequence. If no sign appears, the number is assumed positive. However, the sign may appear anywhere in the specified input field, and NUMGET places it at the beginning of the number. If two signs appear within the same field, the last sign is assumed correct.

When integer data are read in with I format, standard FORTRAN has a number of limitations. The most time-consuming constraint in preparing and keypunching data for a program is the requirement that integers be right-justified and that the sign (if any) precede the number. NUMGET overcomes these restrictions. It is not intended to be an error-correcting routine but rather a convenience and a time-saving routine. Whereas a normal read using integer (I) format of a number such as 77/7 or 77M7 would abort the computer run, NUMGET allows the program to continue, translating the number in each example to 77. Thus the burden of checking the accuracy of the data remains on the analyst who has input the data--but he is able to check an entire run at once, rather than having his run abort on the first illegal field encountered.

The program will scan as many contiguous characters as the calling program has requested (formal parameter NCH), moving down the array NIN (dimensioned as 1 in the subroutine, since the real size of this array is determined by the array size in the calling program, eight characters at a time* until NCH has been satisfied. Since the HIS 6000 has only a 36-bit word and the largest integer that can be translated accurately is $2^{35}-1$, (11 decimal digits), the practical limitation on NCH should be such that, excluding blanks, the translated integer will fit within the word size.

---

* Each word of the HIS 6000 may contain at most six BCD characters.

Perhaps the most common example for NUMGET would be the one where the field width has been defined to be 10 characters (normally read as I10 format). When using NUMGET, the calling program would read the field into an array of size two using a format of A6,A4. NUMGET would be called with NCH set to 10, and all 10 characters would be translated to an integer.

Function NUMGET is illustrated in figure 165.

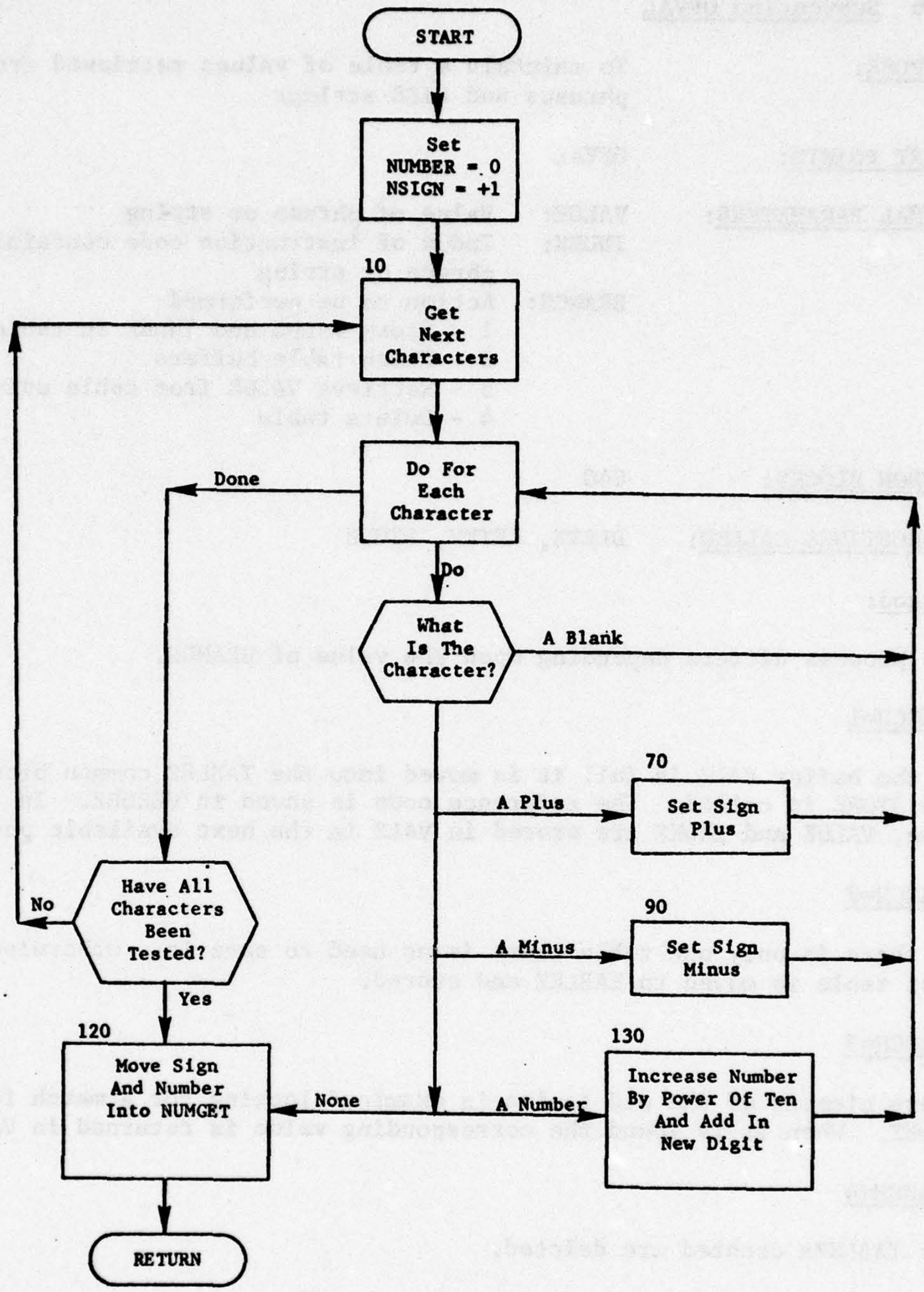


Figure 165. Function NUMGET

9.36 Subroutine OFVAL

PURPOSE: To maintain a table of values retrieved from OF phrases and LIKE strings

ENTRY POINTS: OFVAL

FORMAL PARAMETERS:

VALUE: Value of phrase or string
INDEX: Index of instruction code containing phrase or string
BRANCH: Action to be performed
1 - Store VALUE and INDEX in table
2 - Flush table buffers
3 - Retrieve VALUE from table using INDEX
4 - Delete table

COMMON BLOCKS: C40

SUBROUTINES CALLED: DLETE, RETRV, STORE

Method:

The process differs depending upon the value of BRANCH.

BRANCH=1

If the buffer VALZ is full it is moved into the TABLEZ common block C40 and STORE is called. The reference code is saved in VLHDRZ. In any case, VALUE and INDEX are stored in VALZ in the next available position.

BRANCH=2

If there is only one table there is no need to save it. Otherwise, the last table is moved to TABLEZ and stored.

BRANCH=3

Every element of the old tables is examined looking for a match for INDEX. When it is found the corresponding value is returned in VALUE.

BRANCH=4

Any TABLEZs created are deleted.

Subroutine OFVAL is illustrated in figure 166.

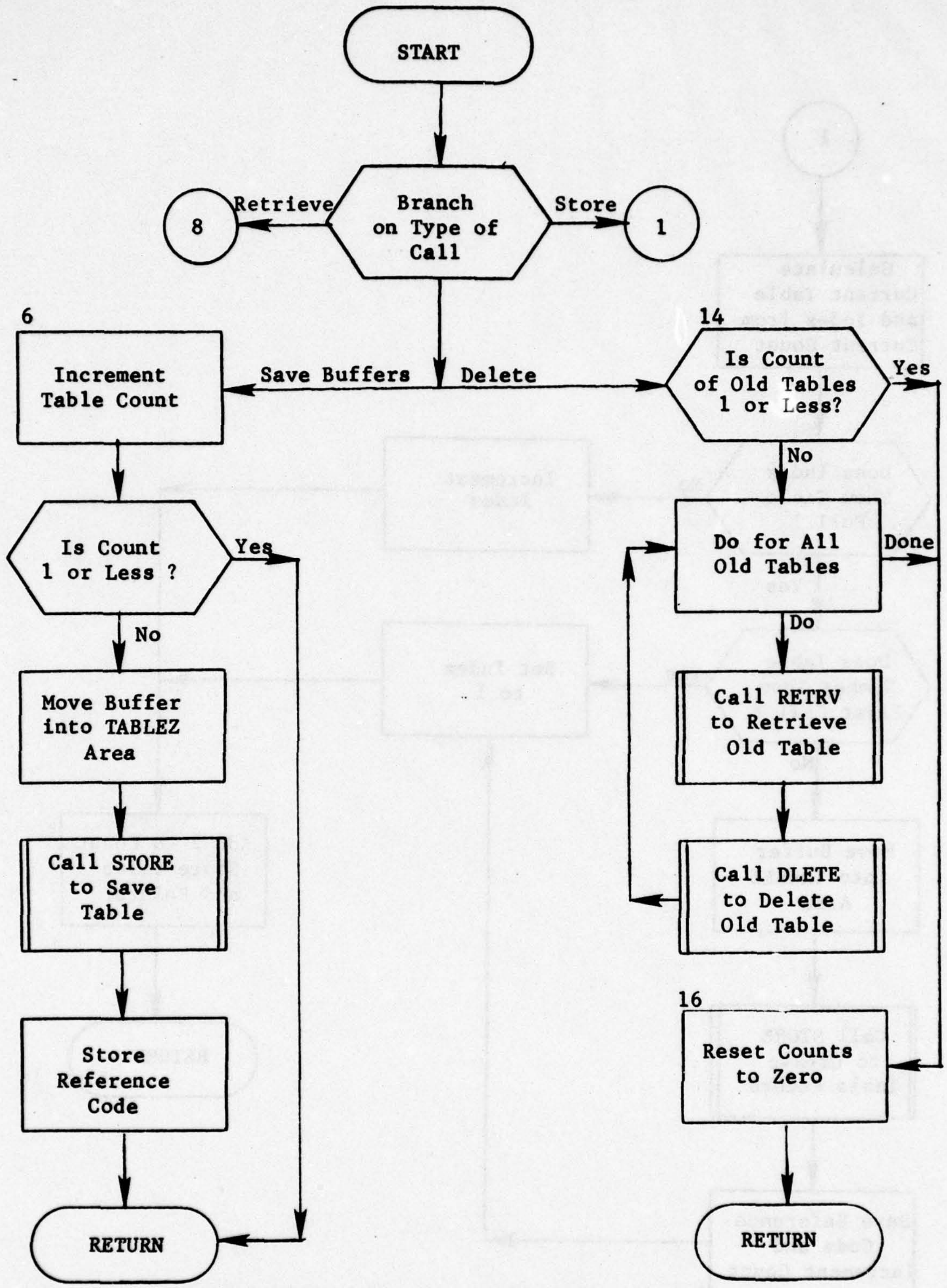


Figure 166. Subroutine OFVAL (Part 1 of 3)

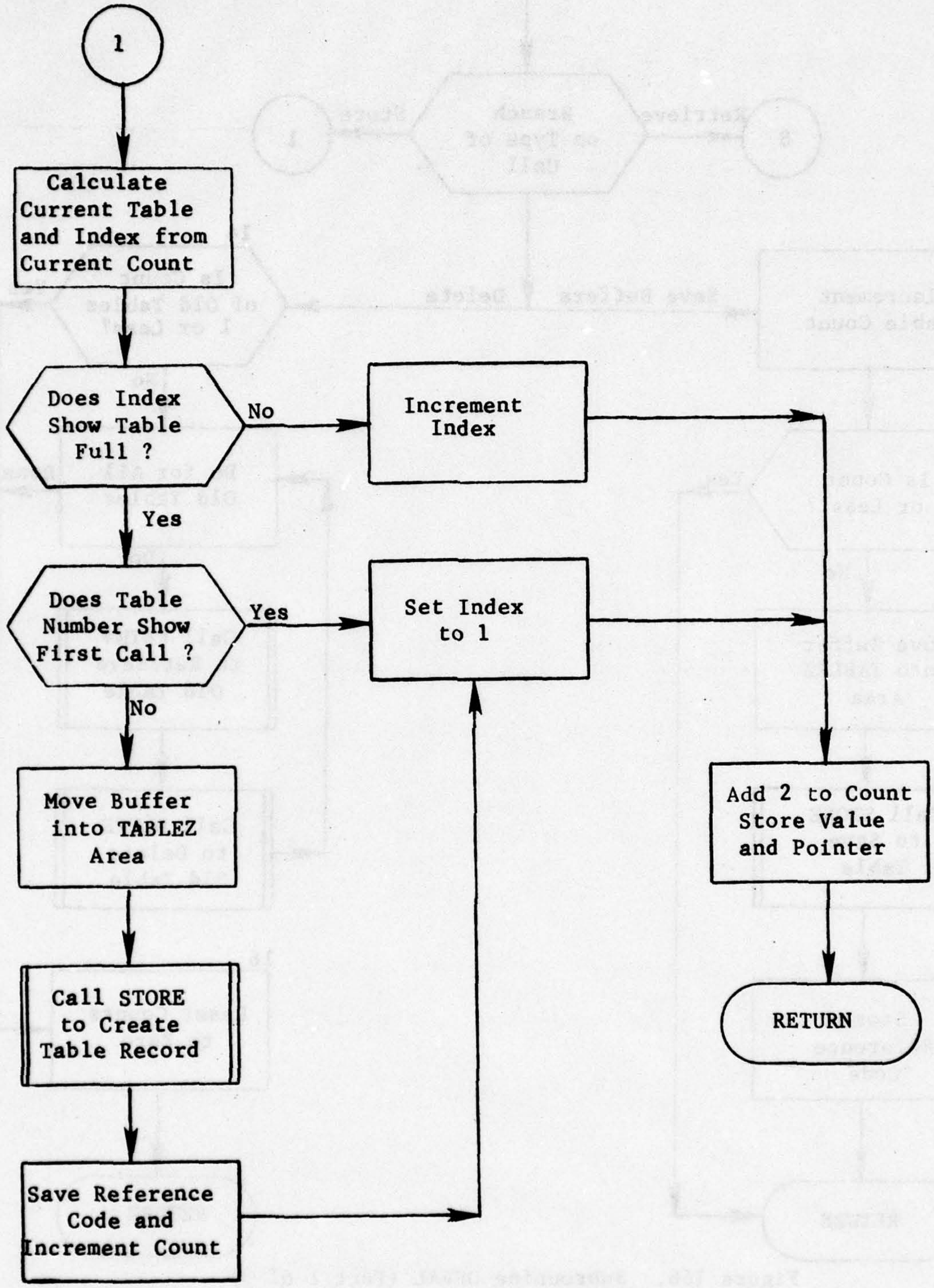


Figure 166. (Part 2 of 3)

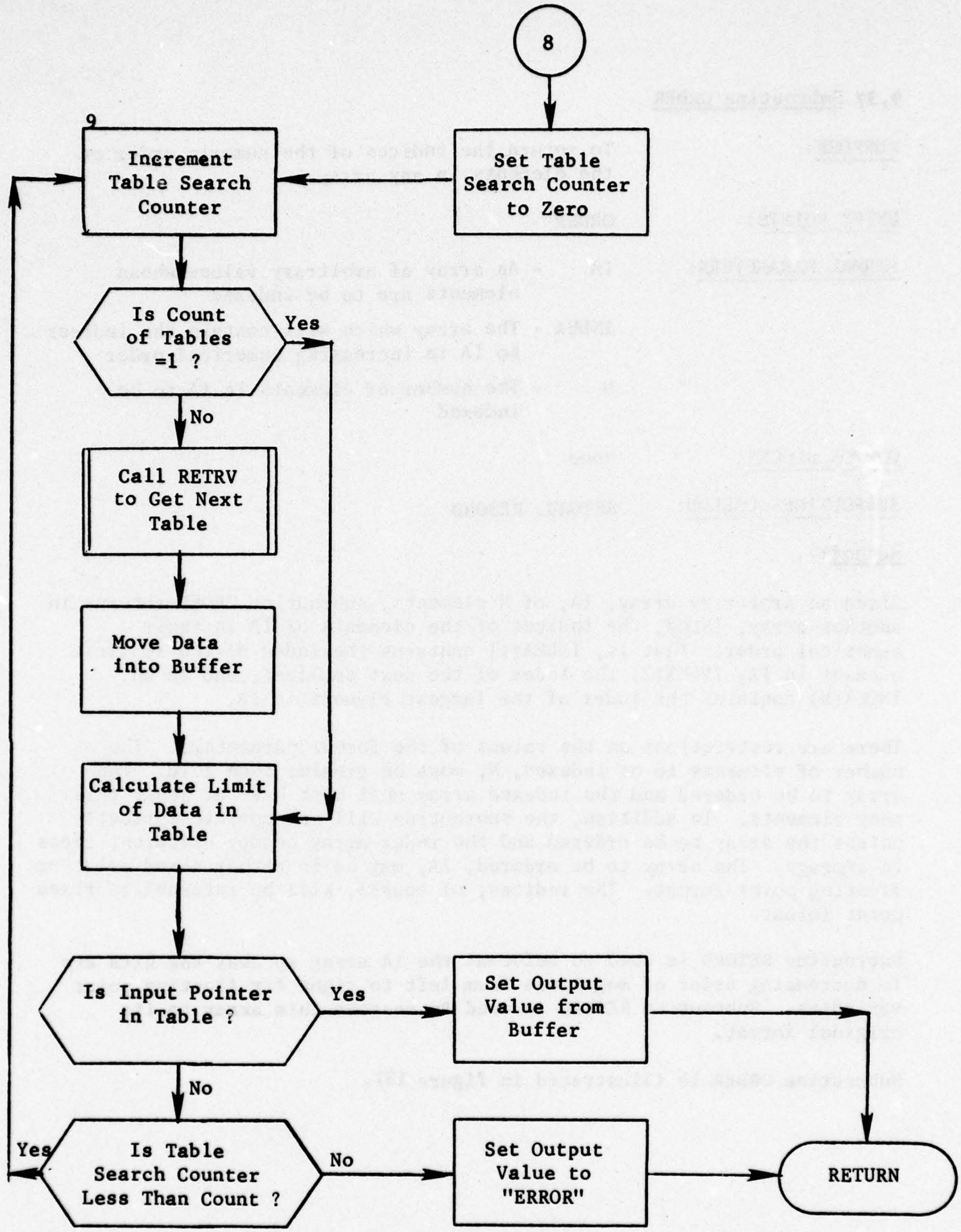


Figure 166. (Part 3 of 3)

9.37 Subroutine ORDER

PURPOSE:	To return the indices of the numeric order of the elements in any array.
ENTRY POINTS:	ORDER
FORMAL PARAMETERS:	IA - An array of arbitrary values whose elements are to be indexed
	INDEX - The array which will contain the indices to IA in increasing numerical order
	N - The number of elements in IA to be indexed
COMMON BLOCKS:	None
SUBROUTINES CALLED:	SETORD, RESORD

Method:

Given an arbitrary array, IA, of N elements, subroutine ORDER returns in another array, INDEX, the indices of the elements of IA in their numerical order. That is, INDEX(1) contains the index of the smallest element in IA, INDEX(2) the index of the next smallest, and so on. INDEX(N) contains the index of the largest element in IA.

There are restrictions on the values of the formal parameters. The number of elements to be indexed, N, must be greater than zero. The array to be ordered and the indexed array must both have at least this many elements. In addition, the subroutine will not operate correctly unless the array to be ordered and the index array occupy different areas in storage. The array to be ordered, IA, may be in either fixed point or floating point format. The indices, of course, will be returned in fixed point format.

Subroutine SETORD is used to reformat the IA array so that the bits are in decreasing order of magnitude from left to right for floating point variables. Subroutine RESORD is used to restore this array to its original format.

Subroutine ORDER is illustrated in figure 167.

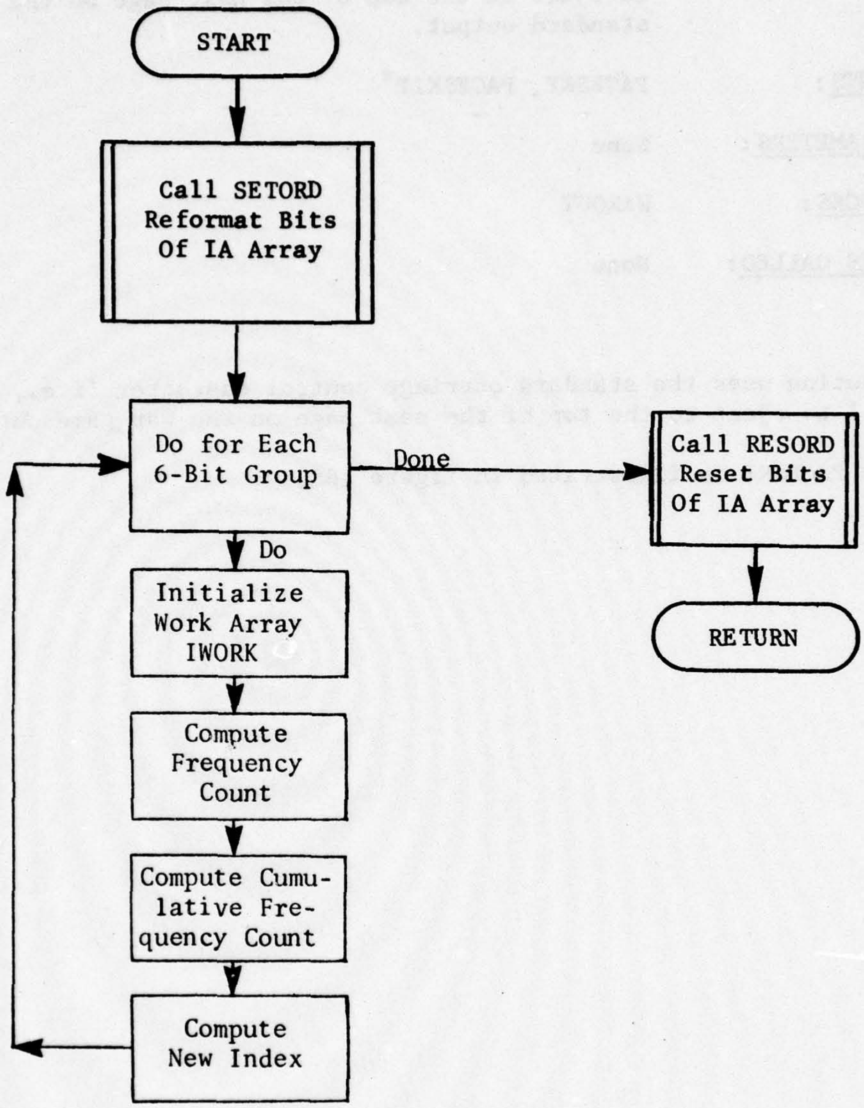


Figure 167. Subroutine ORDER

9.38 Subroutine PAGESKP

PURPOSE:	To eject to the top of the next page on the standard output.
ENTRY POINTS:	PAGESKP, PAGESKIP*
FORMAL PARAMETERS:	None
COMMON BLOCKS:	WAROUT
SUBROUTINES CALLED:	None

Method:

This subroutine uses the standard carriage control character (i.e., 1) in column 1 to eject to the top of the next page on the war game output.

Subroutine PAGESKP is illustrated in figure 168.

* Duplicate entry for PAGESKP.

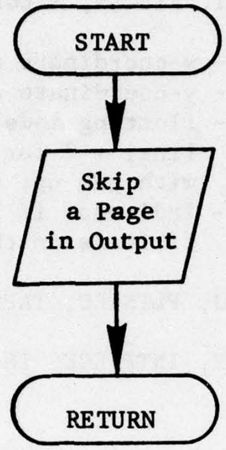


Figure 168. Subroutine PAGESKP

9.39 Subroutine PIECEIT

<u>PURPOSE</u>: To plot those points that fall within the limits of the graph and to write the points which are off the graph on a tape which may be used later to produce labeled plots which can be pieced together to form one large plot.

<u>ENTRY POINTS</u>: PIECEIT, PIECE1, PIECE3, PIECE4

<u>FORMAL PARAMETERS</u>:

X - x-coordinate of point

Y - y-coordinate of point

MODE - Plotting mode indicator (= 1 for dotted line; = 2 for straight line; = 3 to move with pen up; = 4 to change pen color)

IYESN - Indicator if point is on the graph (= 0 if it is on the graph; >0 if it is off)

<u>COMMON BLOCKS</u>: PLTPROJ, PLTSPEC, TAPES, SNDMIN, ISUP

<u>SUBROUTINES CALLED</u>: DOTLINE, INTPIECE, ISOFF, NEWPEN*, PLOT*

<u>Method</u>:

When PIECEIT is called, the mode indicator is tested. If it equals 4, only the color of the pen is changed and the PIECTAPE is written with that information. Then control is returned to the calling program. For any other mode subroutine ISOFF is called to determine if the point (X,Y) falls within the limits of the graph. If IYESN was returned with a value greater than zero, one or both coordinates of point (X,Y) are off the graph. No plotting is performed in this case. The point coordinates and mode indicator are output on the PIECTAPE and saved for the next call on PIECEIT.

If IYESN was returned with a value of zero, point (X,Y) is on the graph. The information saved from the previous point is tested if it was on the graph or not. If it was off, subroutine INTPIECE is called to position the pen at the point along the edge of the graph where the line connecting the previous and present points would intersect it. The mode indicator is tested and if a dotted line is requested, subroutine DOTLINE is called; otherwise, the PLOT routine is called to draw a line or move the pen from its present position to point (X,Y). The PIECTAPE is always written to keep all the information for further plots.

---

* These are standard plotting routines and not documented in this manual. They are only included for completeness.

At entry PIECE1, routine NEWPEN is called to change the pen color and write the projection parameters on the PIECTAPE.

At entry PIECE3, the number of points off the graph and the actual values of the x- and y-coordinates at the points with the minimum x value and the minimum y value are printed. The PIECTAPE is written to indicate the end of a plot.

At entry PIECE4, the projection parameters and an end-of-file mark are written on the PIECTAPE. Then the tape is rewound.

Subroutine PIECEIT is illustrated in figure 169.

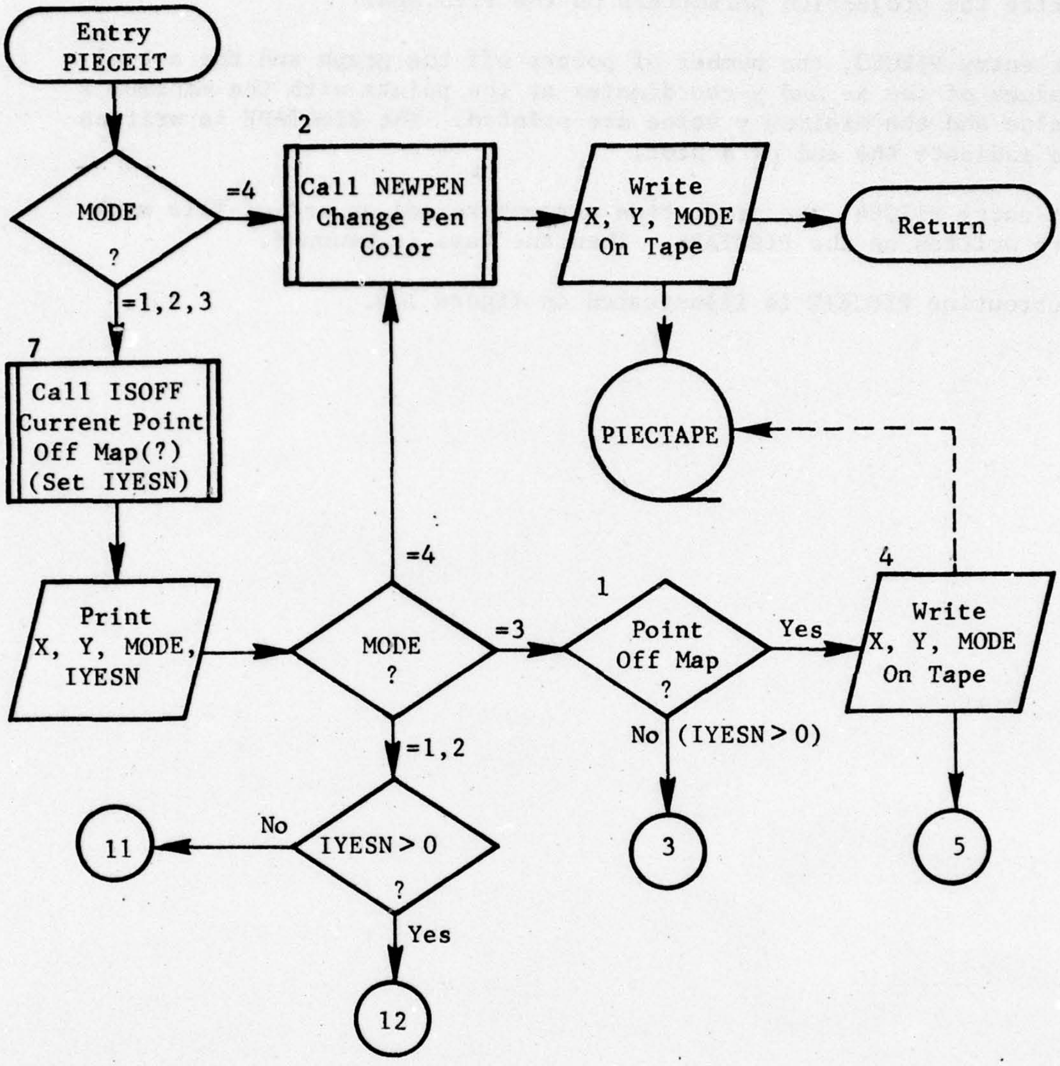


Figure 169. Subroutine PIECEIT (Part 1 of 3)

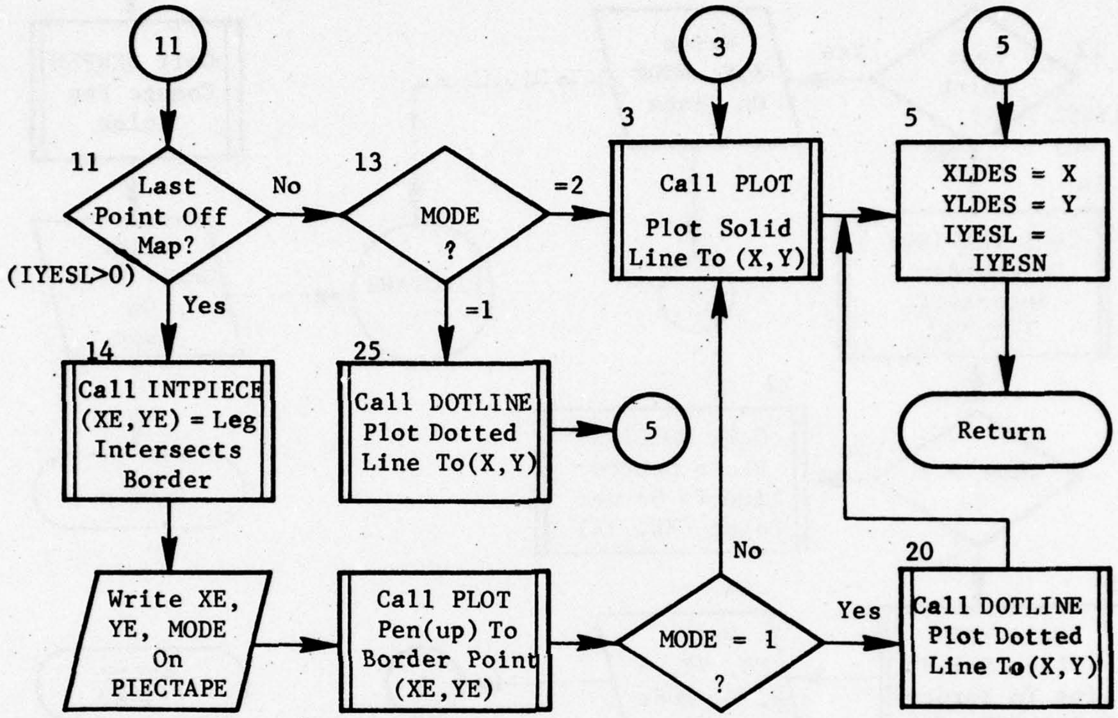


Figure 169. (Part 2 of 3)

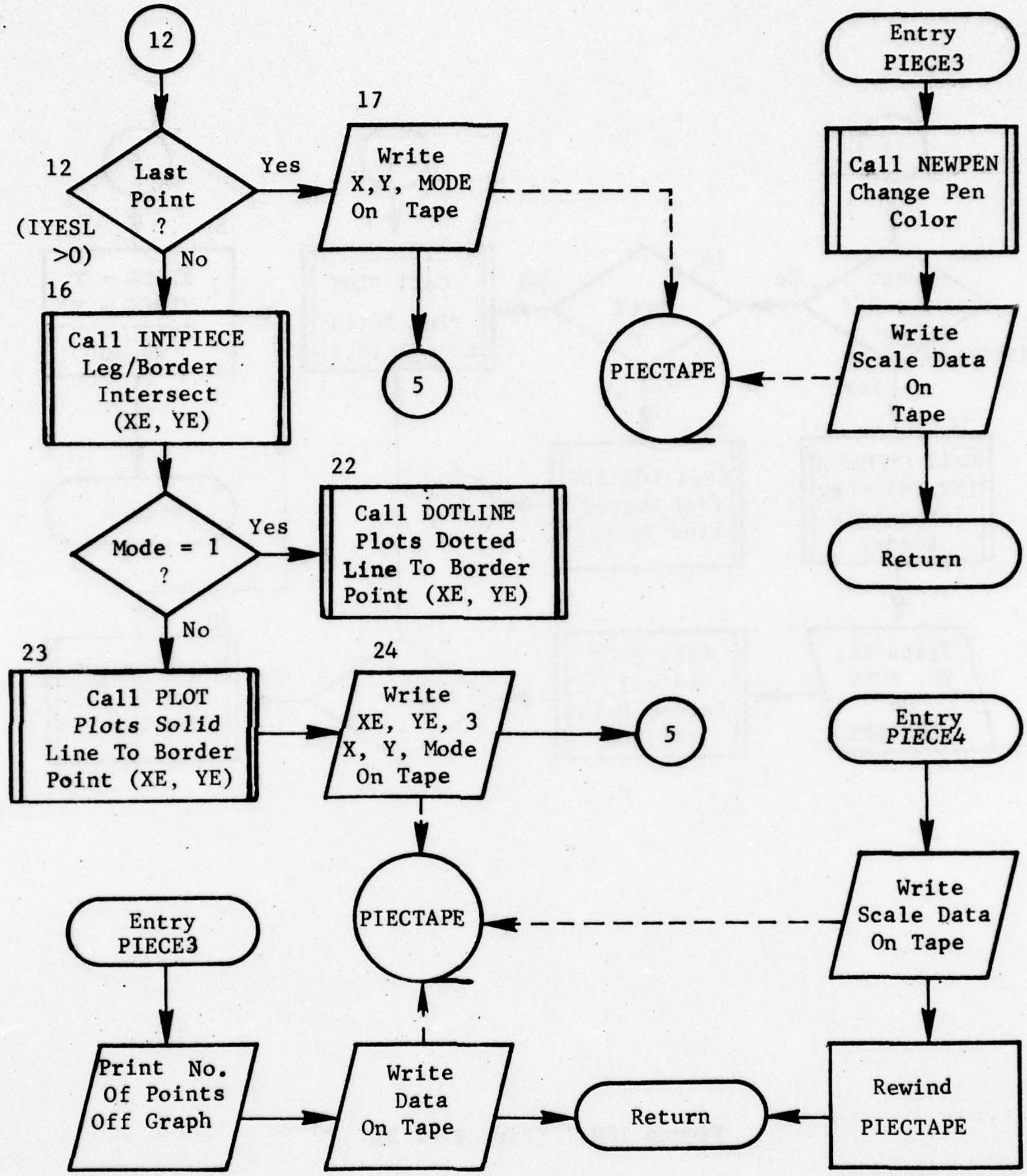


Figure 169. (Part 3 of 3)

9.40 Subroutine PIECENUM

PURPOSE: To write the labels on a plot.

ENTRY POINTS: PIECENUM

FORMAL PARAMETERS:

X - Value of x-coordinate
Y - Value of y-coordinate
MODE - Plotting indicator to be used for call on subroutine PIECEIT
NUM - Indicator (= 0 to plot a number; = 1 to plot a symbol)
HEIGHT - Size of label
BCD - Point number or symbol
THETA - Angle in which label is to be plotted
NN - Indicator for plotting a number or a symbol (if NUM = 0 and NN = -1, the decimal point is plotted, NN ≥1 NN digits to the right of the decimal point are plotted; if NUM = 1 and NN ≥0 NN characters from variable BCD will be plotted; NN = -1 the centered character defined by BCD will be plotted)
IYESN - Indicator if point is on the graph (= 0 if point is on the graph, >0 if point is off)

COMMON BLOCKS: TAPES, ISUP

SUBROUTINES CALLED: NUMBER*, PIECEIT, SYMBOL*

Method:

First it has to be determined if the label will fall within the limits of the graph. Subroutine PIECEIT is called to obtain a value for the indicator IYESN. If IYESN equals zero, which means that the label is on the graph, the routine NUMBER or SYMBOL is called as indicated by NUM, to plot a number or a symbol. If the label is not on the graph, the PIECTAPE is written with all the label information.

Subroutine PIECENUM is illustrated in figure 170.

---

* These are standard plotting routines and not documented in this manual. They are only included for completeness.

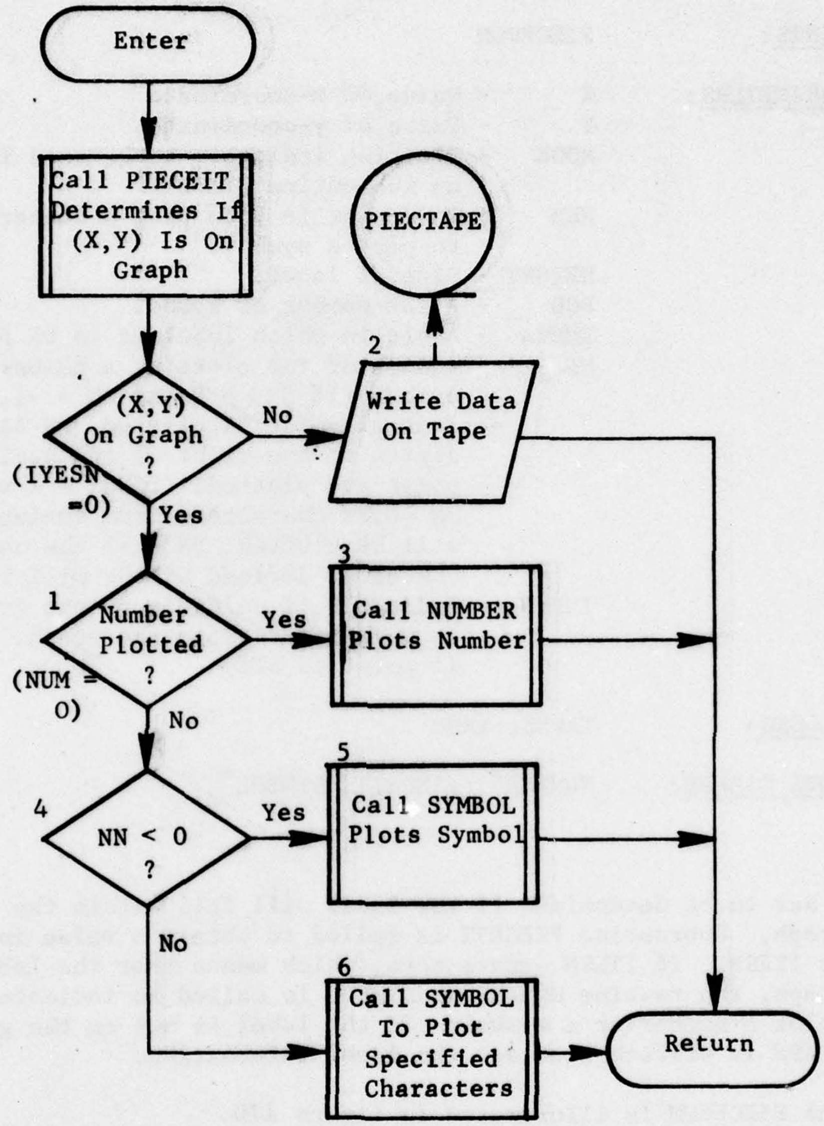


Figure 170. Subroutine PIECENUM

9.41 Subroutine PLTTIC

PURPOSE: To perform the actual plotting of the ticmarks.

ENTRY POINTS: PLTTIC

FORMAL PARAMETERS:

NTIC - Number of ticmarks
TICX - Array of x-coordinates for ticmarks
TICY - Array of y-coordinates for ticmarks
XO, YO - Coordinates of point where pen is to be positioned after ticmarks have been plotted

COMMON BLOCKS: None

SUBROUTINES CALLED: PLOT*

Method:

For all ticmarks the routine PLOT is called; first to position the pen at the starting point for each ticmark; then to connect the three points that outline the ticmark. Before returning control to the calling program, the pen is positioned at the point (XO,YO).

Subroutine PLTTIC is illustrated in figure 171.

---

* This is a standard plotting routine and not documented in this manual. It is only included for completeness.

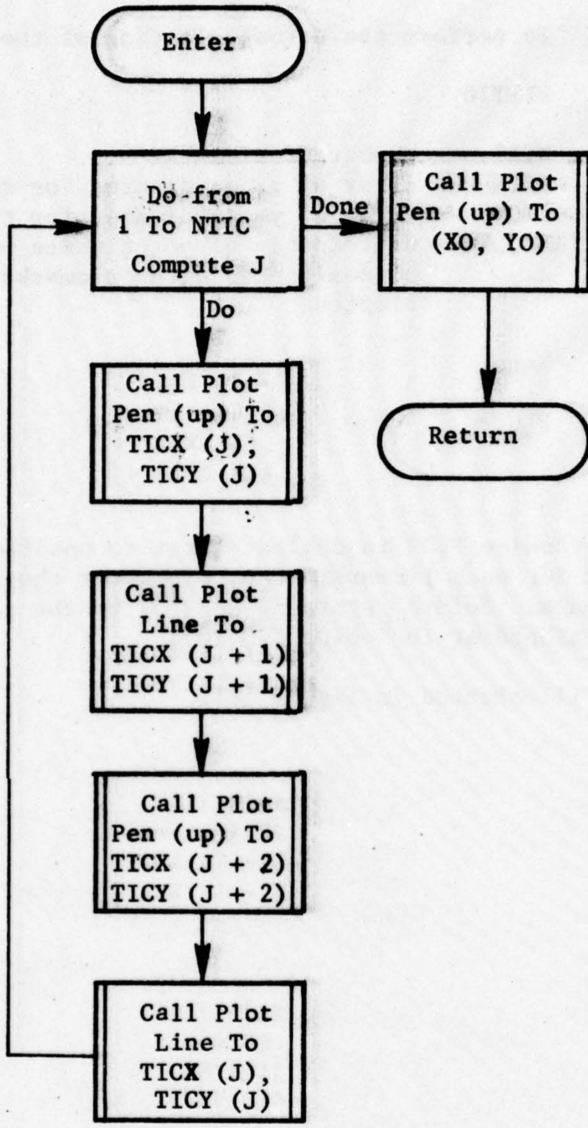


Figure 171. Subroutine PLTTIC

9.42 Subroutine PRIMHD

PURPOSE:	To find the primary header	
ENTRY POINTS:	PRIMHD	
FORMAL PARAMETERS:	RECTYP:	Input record type number output as record type number of input's primary header
COMMON BLOCKS:	C20, C30	
SUBROUTINES CALLED:	NEXTTT	

Method:

The process involves altering the value of RECTYP in an iterative manner until the exit condition occurs. The first step of the iteration is find the corresponding INDRCT record on the RCTYP chain. Then if RECTYP is less than or equal to 30 (a header) the subroutine exits. Otherwise NEXTTT is called on the IRDET chain and RECTYP is set equal to the master record number.

Subroutine PRIMHD is illustrated in figure 172.

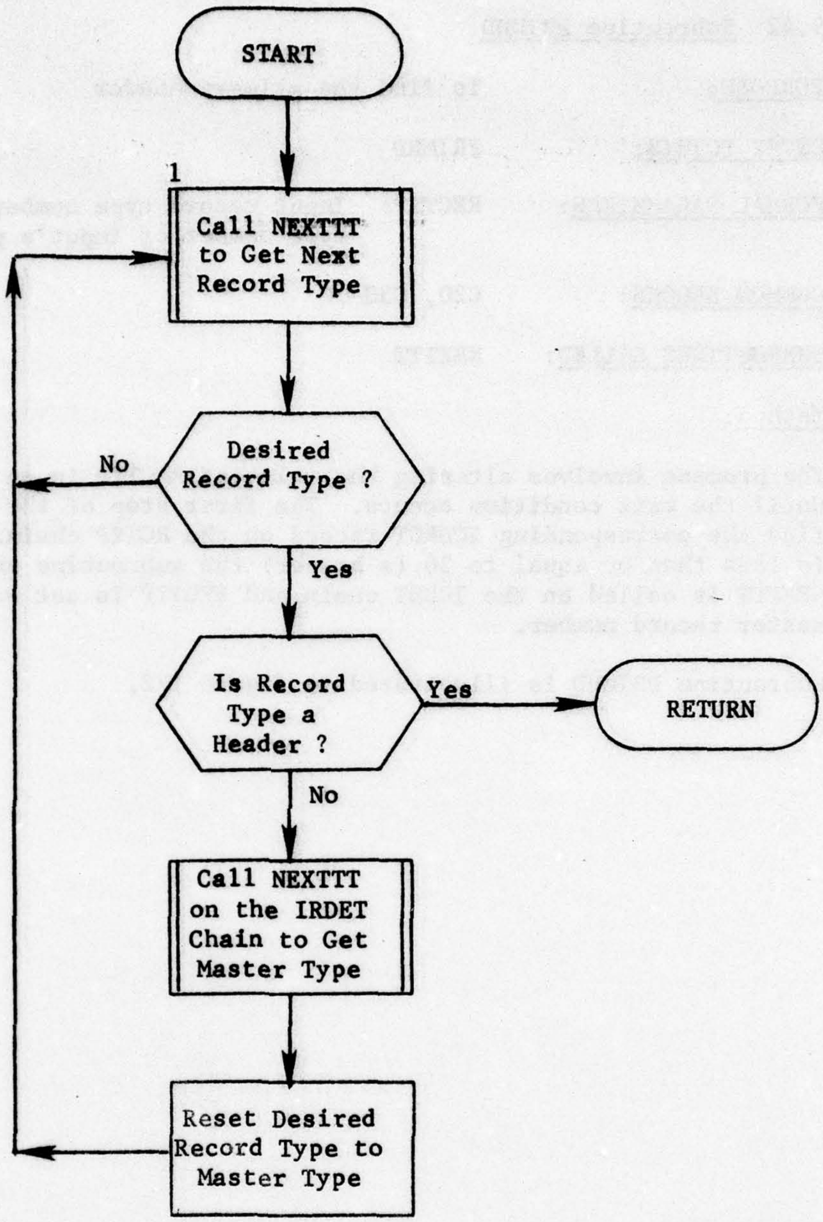


Figure 172. Subroutine PRIMHD

9.43 Subroutine PSREC

PURPOSE:	To create, maintain, sort and retrieve print/sort records
ENTRY POINTS:	PSNXT, PSPUT, PSREC, PSRWD, XSORT
FORMAL PARAMETERS:	None
COMMON BLOCKS:	PSCOM, SORSCH
SUBROUTINES CALLED:	SORTIT
CALLED BY:	BLDOTH, PRINT

Method:

Entry Point PSREC

First RANSIZ is called. Then the length of the sort scheme (LENSRT) is determined and from it the length of the sort record (LENREC) calculated. Finally the record count (INDEX), file mark check (FYLMRK) and file unit (UNIT) are initialized.

Entry Point PSPUT

This entry adds records to the file. For each record INDEX is incremented and the index sequential record is stored. If LENSRT is greater than zero the sort record is also built as follows. The sort scheme is executed. For each instruction in the scheme the indicated record element is retrieved, converted according to its mode and stored in the sort record. Descending keys are complemented before they are stored. Then the sort keys plus INDEX are written onto UNIT.

Entry Point PSRWD

If LENSRT is greater than zero FYLMRK is checked and if false an end of file is written on UNIT. UNIT is then rewound. If LENSRT is zero, INDEX is set to zero.

Entry Point XSORT

A call is made to SORTIT and a RETURN is executed.

Entry Point PSNXT

If LENSRT is not zero the next record is read from UNIT. The value for INDEX is then used to retrieve the appropriate index sequential record. If LENSRT is zero, INDEX is incremented and the appropriate index sequential record retrieved.

Subroutine PSREC is illustrated in figure 173.

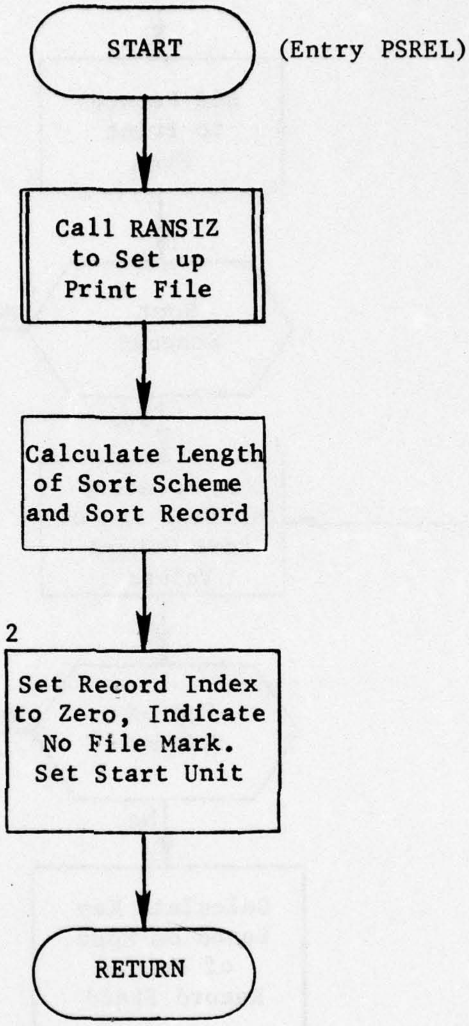
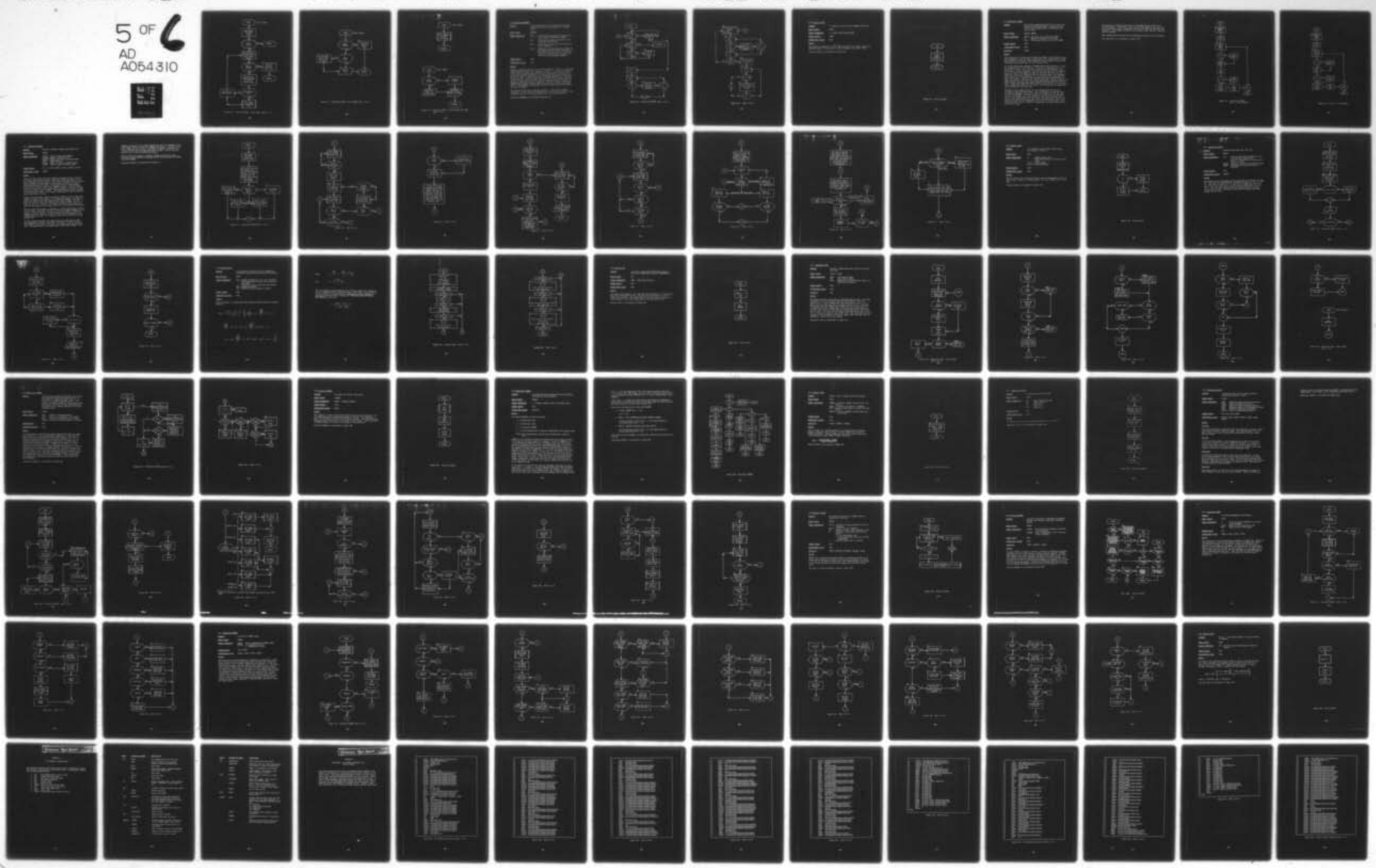


Figure 173. Subroutine PSREC: Entry PSREC (Part 1 of 4)

AD-A054 310 COMMAND AND CONTROL TECHNICAL CENTER WASHINGTON D C F/G 15/7
THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK) PROGR--ETC(U)
JUN 77 D J SANDERS, P F MAYKRANTZ, J M HERRIN
UNCLASSIFIED CCTC-CSM-MM-9-77-V1-PT-2 SBIE-AD-E100 052 NL

5 OF 6
AD
A054310



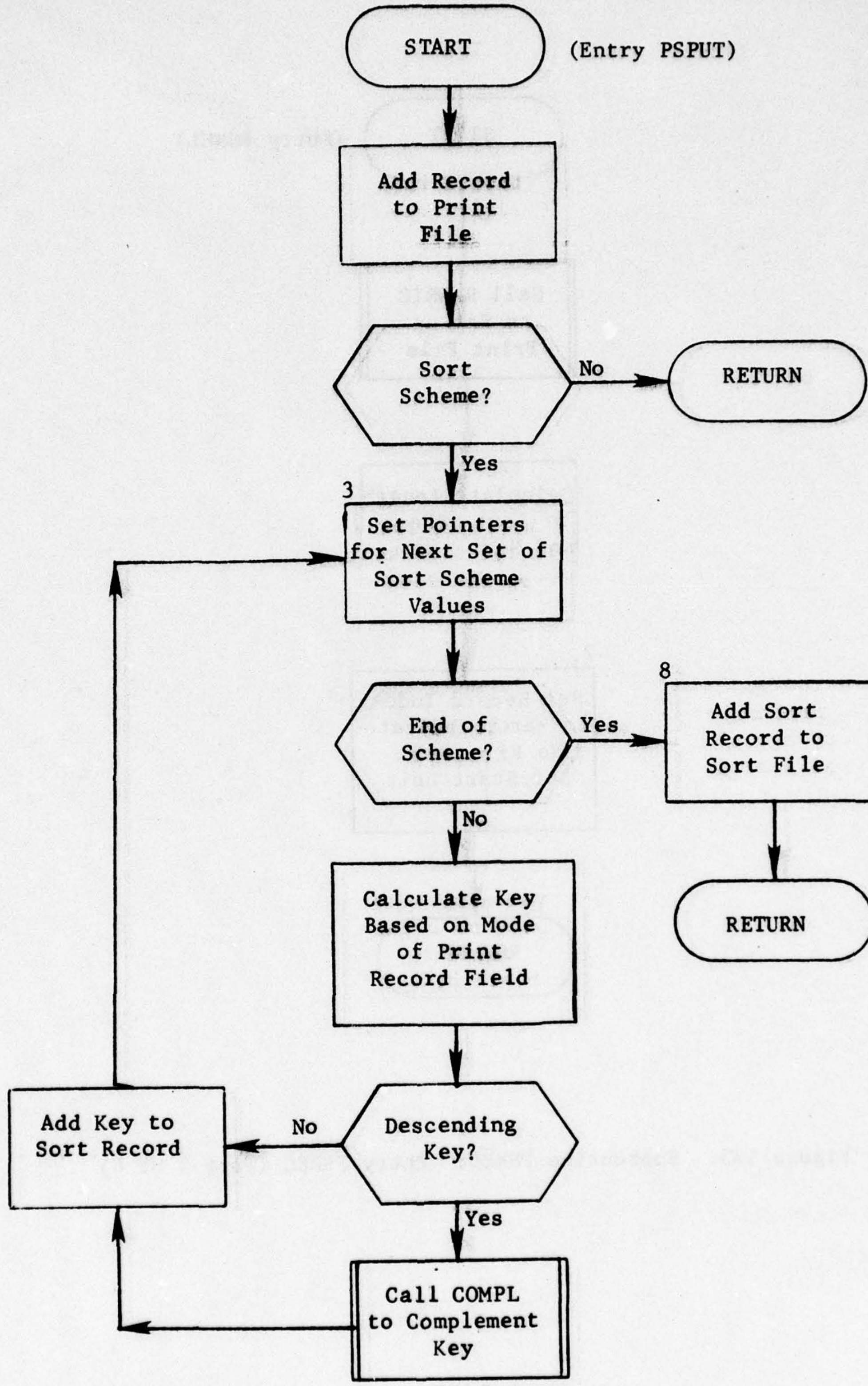


Figure 173. Subroutine PSREC: Entry PSPUT (Part 2 of 4)

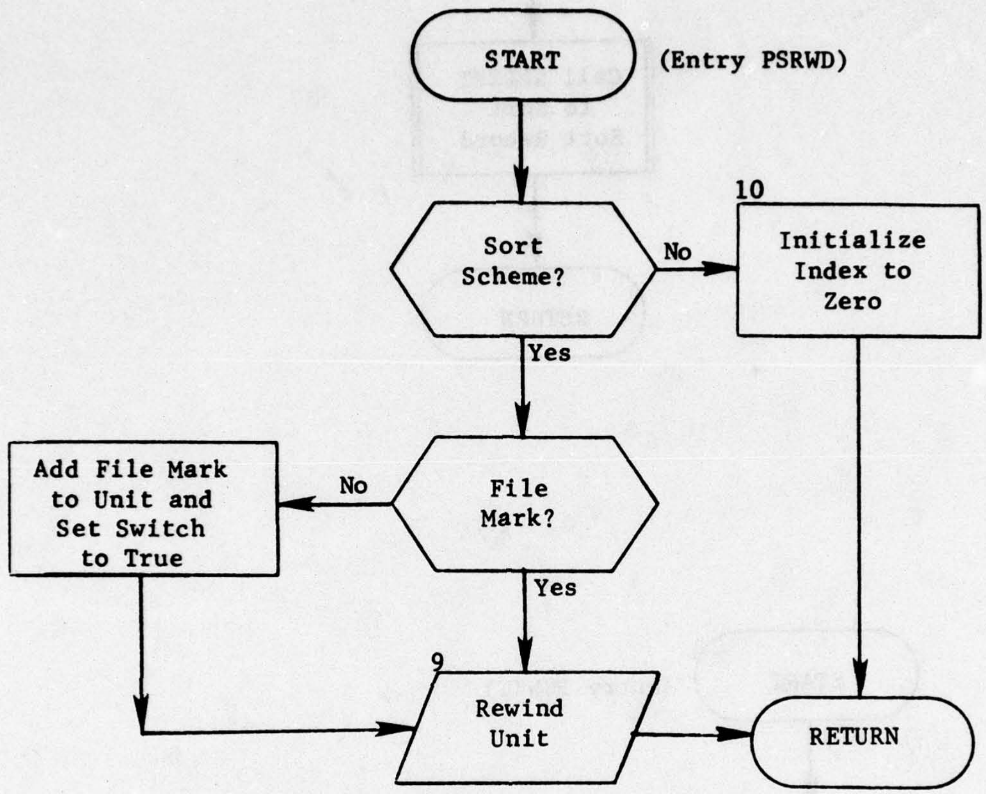


Figure 173. Subroutine PSREC: Entry PSRWD (Part 3 of 4)

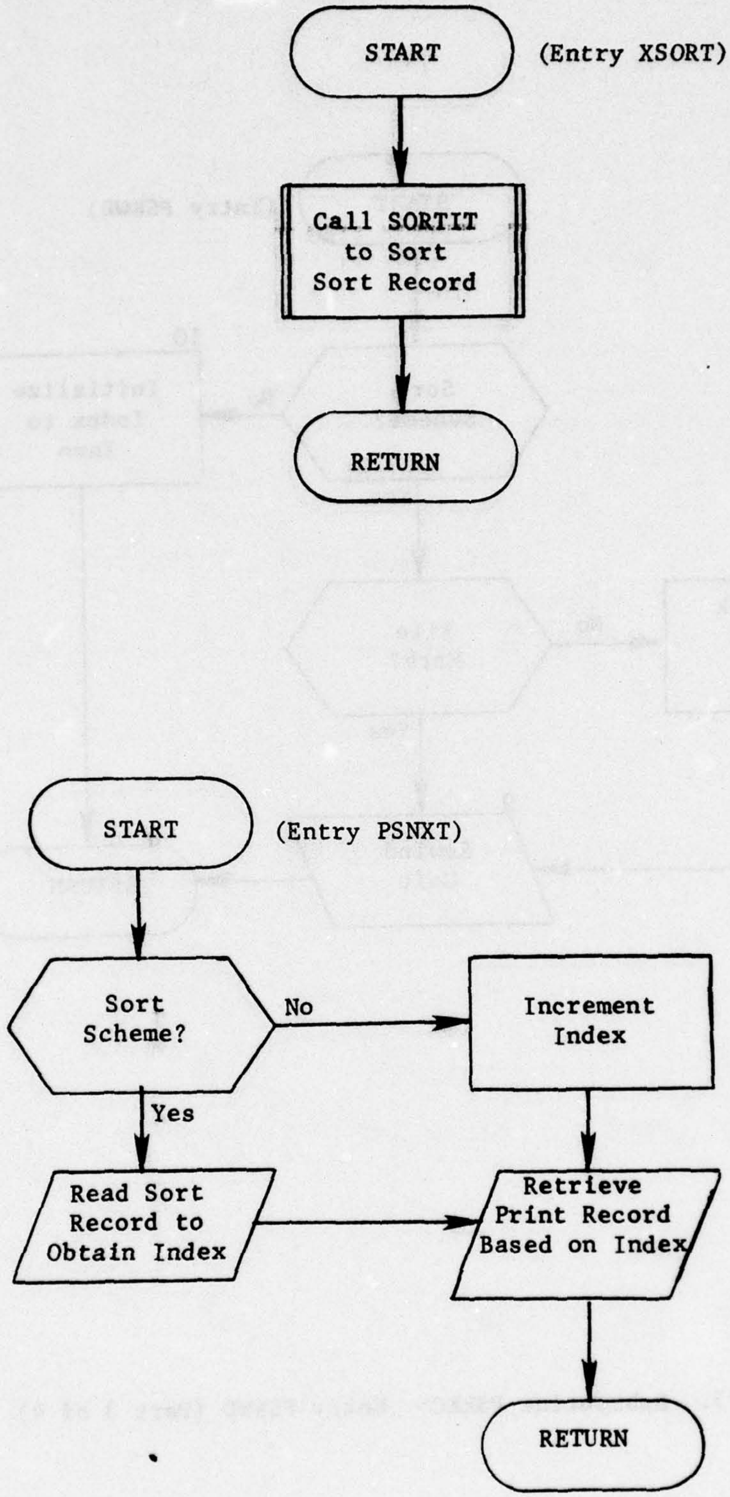


Figure 173. Subroutine PSREC: Entries XSORT and PSNXT (Part 4 of 4)

9.44 Subroutine REORDER

PURPOSE:	To rearrange from one to seven arrays into the sequence specified by the elements of an index array.	
ENTRY POINTS:	REORDER	
FORMAL PARAMETERS:	ISEQ	- (Fixed point) sequence key array of the type produced by subroutine ORDER
	NEL	- (Fixed point) number of elements to be reordered in each array
	NAR	- (Fixed point) number of arrays which are to be reordered
	L1-L7	- (Fixed point or floating point) names of the arrays which are to be reordered; if the number of these arrays is less than seven, the remaining positions must be filled by trailing dummy arguments
COMMON BLOCKS:	None	
SUBROUTINES CALLED:	None	

Method:

Subroutine REORDER operates in the following manner. First, it stores one element from each array in a temporary location. It then reads from the array ISEQ the element which should go in that position (which may now be considered empty) and moves it, filling that position and creating a new blank. This is repeated for the corresponding element of each array being reordered. Each new blank spot is filled with its proper contents as soon as its original contents have been moved, until the element currently in temporary storage is required. When this happens, subroutine REORDER finds another element which is not already in its sequence and puts it into temporary storage and continues as before until no elements are out of sequence.

The contents of the array ISEQ are returned to the calling program unchanged, so that subroutine REORDER may be called again, using the same sequence key array if more than seven arrays are to be reordered.

Subroutine REORDER is illustrated in figure 174.

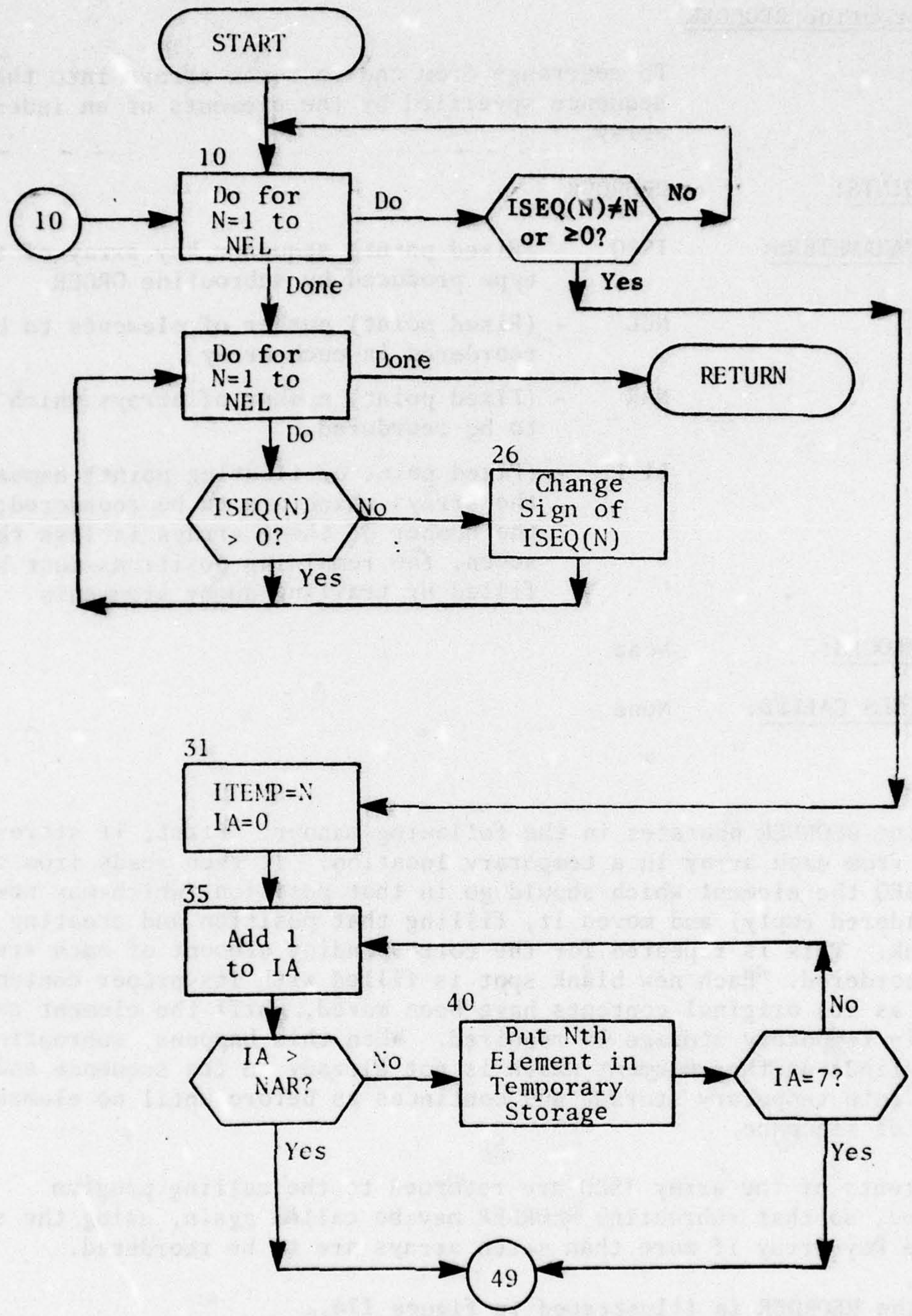


Figure 174. Subroutine REORDER (Part 1 of 2)

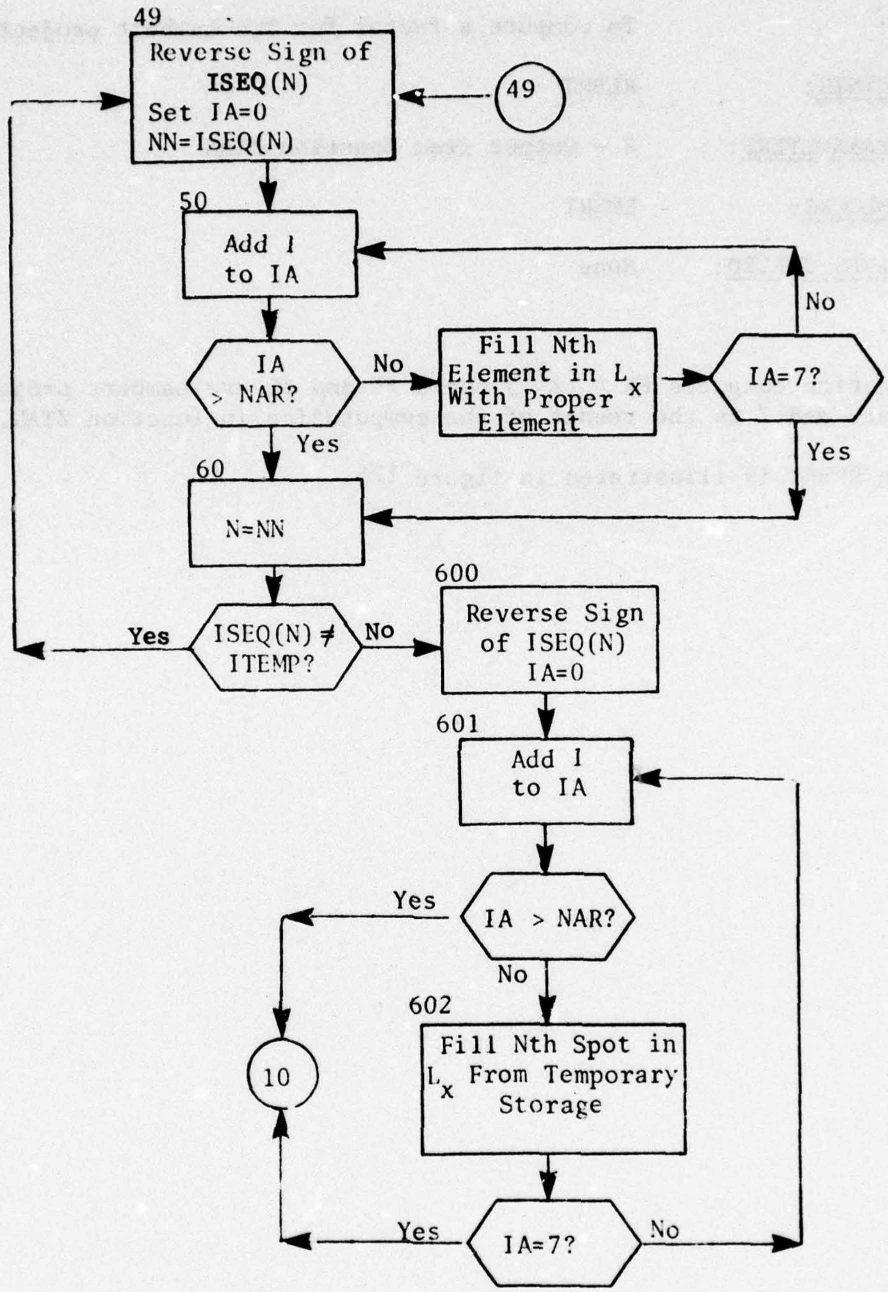


Figure 174. (Part 2 of 2)

9.45 Function RLBRT

PURPOSE: To compute a factor for the Lambert projection.

ENTRY POINTS: RLBRT

FORMAT PARAMETERS: Z - Output from function ZTAN

COMMON BLOCKS: LMBRT

SUBROUTINES CALLED: None

Method:

This function computes FK * (Z^{FL}) where FK and FL are Lambert projection parameters and Z is the result of the computation in function ZTAN.

Function RLBRT is illustrated in figure 175.

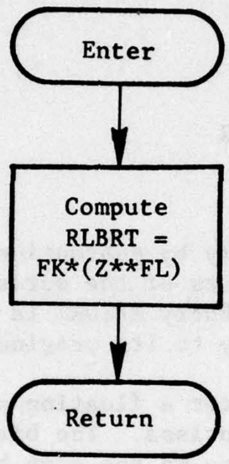


Figure 175. Function RLBRT

9.46 Subroutine SETORD

PURPOSE:	This routine manipulates the bits of words to be ordered by subroutine ORDER to create a string of bits whose significance is decreasing from left to right.
ENTRY POINTS:	SETORD, RESORD
FORMAL PARAMETERS:	ARRAY - The array to be ordered by ORDER N - Number of elements in the array ISEQ - The sequence array to be set by ORDER
COMMON BLOCKS:	None
SUBROUTINES CALLED:	None
CALLED BY:	ORDER

Method:

This subroutine is called only by subroutine ORDER. Entry SETORD is used before ordering to set the bits of the words to be ordered into a format more amenable to ordering. Entry RESORT is used to restore the bit configuration of the input array to its original value.

On the HIS 6000 series computer a floating point word consists of a nine-bit exponent and a 27-bit mantissa. The bits are numbered zero to 35 going left to right. Bit zero is the sign bit for the exponent (zero for positive, one for negative); bits one through eight are the exponent value; bit nine is the mantissa sign bit (zero for positive, one for negative); and bits 10 through 35 are the mantissa value. For floating point value, the most significant bit is bit nine which determines the sign of the value. Then bits zero through eight are the next significant, followed by 10 through 35 as least significant. However, subroutine ORDER requires that bit significance decrease monotonically from left to right. Subroutine ORDER also assumes that a value of one for a bit signifies a higher value than the value zero. This convention is contrary to the HIS convention for sign bits.

Therefore, entry SETORD revises the bit configuration of the word according to the following scheme. Bits 10 through 35 of the new word (at exit from SETORD) are equal to bits 10 through 35 of the old word (at entry to SETORD). Bits two through nine of the new word are equal to bits one through eight of the old word. Bit one of the new word is equal to the complement of bit zero of the old word and bit zero of the new word is equal to the complement of bit nine of the old word. This scheme produces a word in which bit significance decreases monotonically from left to right.

This process is carried on for each of the N words in the array to be ordered (ARRAY). Array ISEQ is used by subroutine ORDER so that the bits are in decreasing order of magnitude from left to right for floating point variables. Entry SETORD initializes this array so that ISEQ(J)= J for the first N elements.

Entry RESORD merely reverses the bit manipulation process of entry SETORD.

This subroutine is illustrated in figure 176.

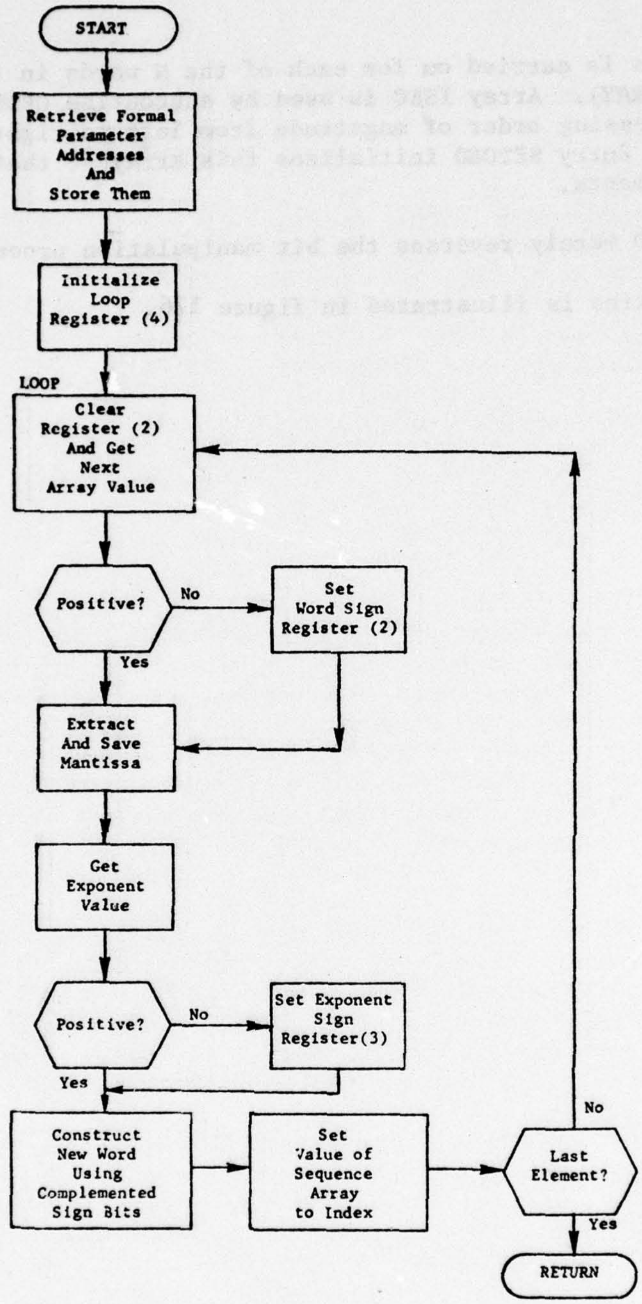


Figure 176. Subroutine SETORD
Part I: Entry SETORD

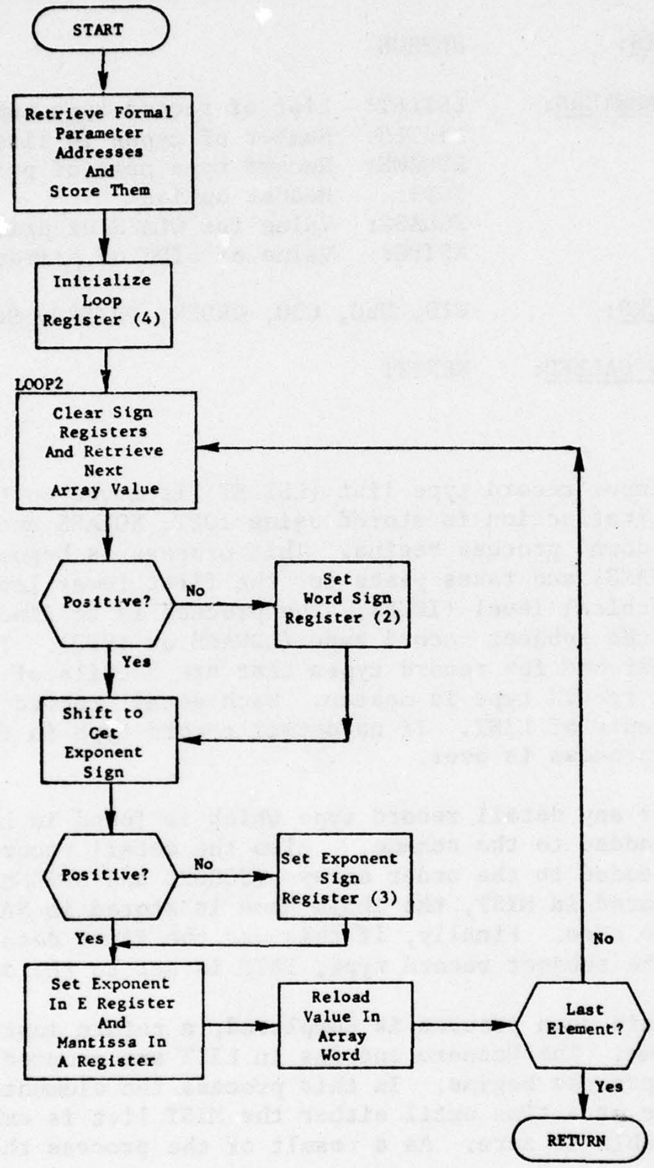


Figure 176. Part II: Entry RESORD

9.47 Subroutine SETSCH

PURPOSE: Builds a retrieval scheme (see section 4.4)

ENTRY POINTS: SETSCH

FORMAL PARAMETERS:

LSTLST: List of record type numbers
LISTLN: Number of types in list
HDNAME: Record type name of primary header
IOPT: Header option
XCLASS: Value for CLASS of primary header
XSIDE: Value of SIDE of primary header

COMMON BLOCKS: C10, C20, C30, ORDER, PRINSP. SCHEME, SCRATCH

SUBROUTINES CALLED: NEXTTT

Method:

First the input record type list (LSTLST) is moved to LIST. Then the Get Header instruction is stored using IOPT, XCLASS and XSIDE. Now the 'chain down' process begins. This process is begun with the primary header (HDNAME) and takes place for the first lower level record of each hierarchical level (IRTP). The process is to find the INDRCT record for the subject record type (HDNAME or IRTP). Then the IRMAST chain is searched for record types that are details of chains for which the subject record type is master. Each detail record type is compared to the contents of LIST. If no detail record type is found in LIST the chain down process is over.

However, for any detail record type which is found in LIST a chain next command is added to the scheme. Also the detail record type name and number are added to the order arrays (SCHORD and SORDNM). The LIST entry is stored in MIST, the chain name is stored in NAMZ, and the LIST entry set to zero. Finally, if this was the first detail record type found for the subject record type, IRTP is set to the detail record type.

When the chain down process is completed, a return instruction is added to the scheme. The Nonzero entries in LIST are counted (NONZER) and the 'chain up' process begins. In this process the elements of MIST are examined one at a time until either the MIST list is exhausted or the NONZER variable is zero. As a result of the process the MIST list may be added to.

For each element examined, the INDRCT record is found and the IRDET chain searched for record types which are masters of chains of which the examined record type is a detail. Each master type is compared to LIST. Any match which is found generates a head chain command. This

command is inserted in the scheme immediately after the command for the chain (NAMZ) which retrieves the examined record type. The new record type's number is also inserted in SCHORD and SORDNM. Furthermore, it is added to MIST and the chain name added to NAMZ. Finally the LIST entry is set to zero and NONZER decremented.

When the chain up process is complete, SETSCH prepares the scheme execution process (GETNXT) by setting POINT=1 and retrieving the INDRCT record for HDNAME.

Subroutine SETSCH is illustrated in figure 177.

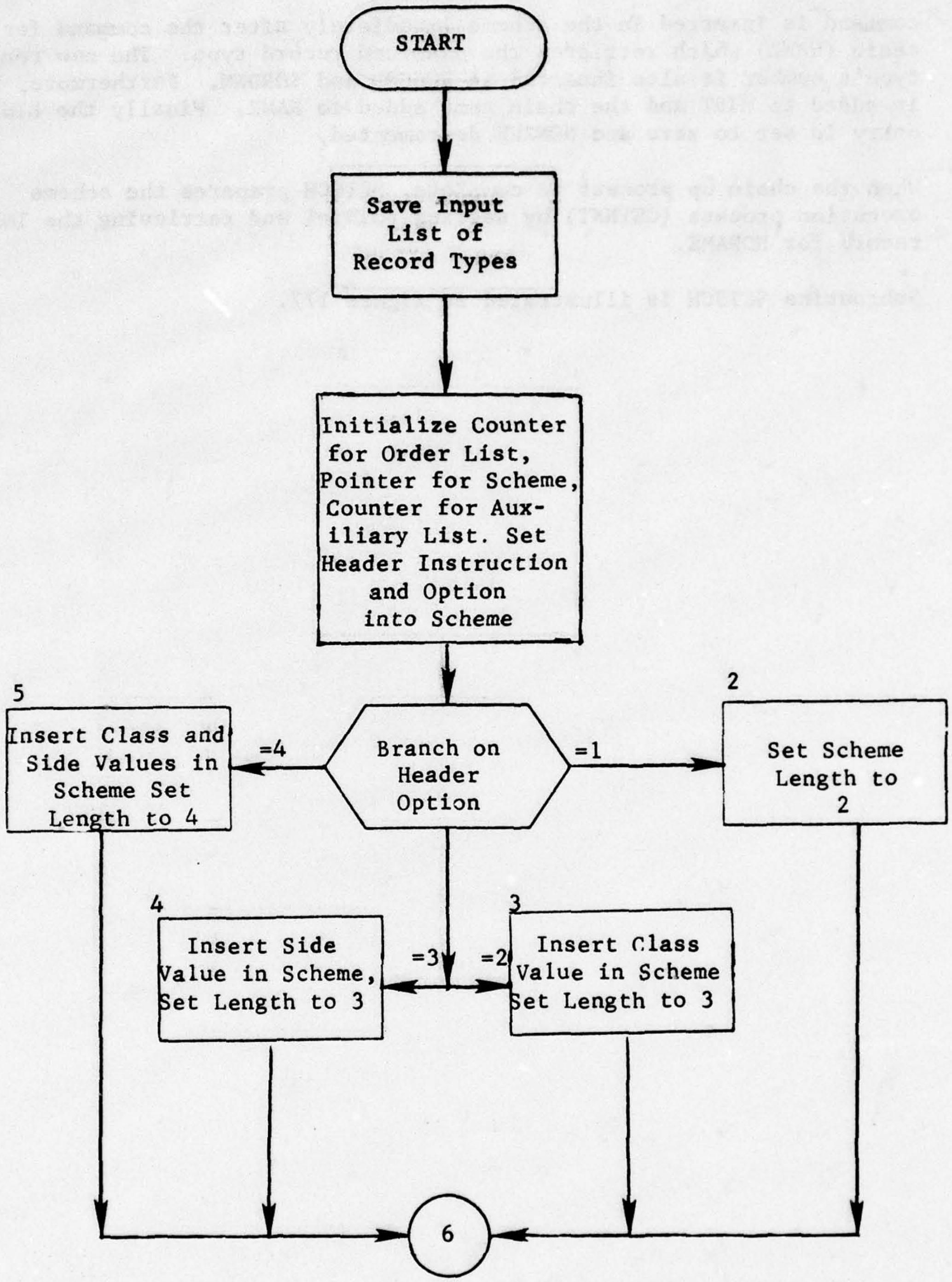


Figure 177. Subroutine SETSCH (Part 1 of 8)

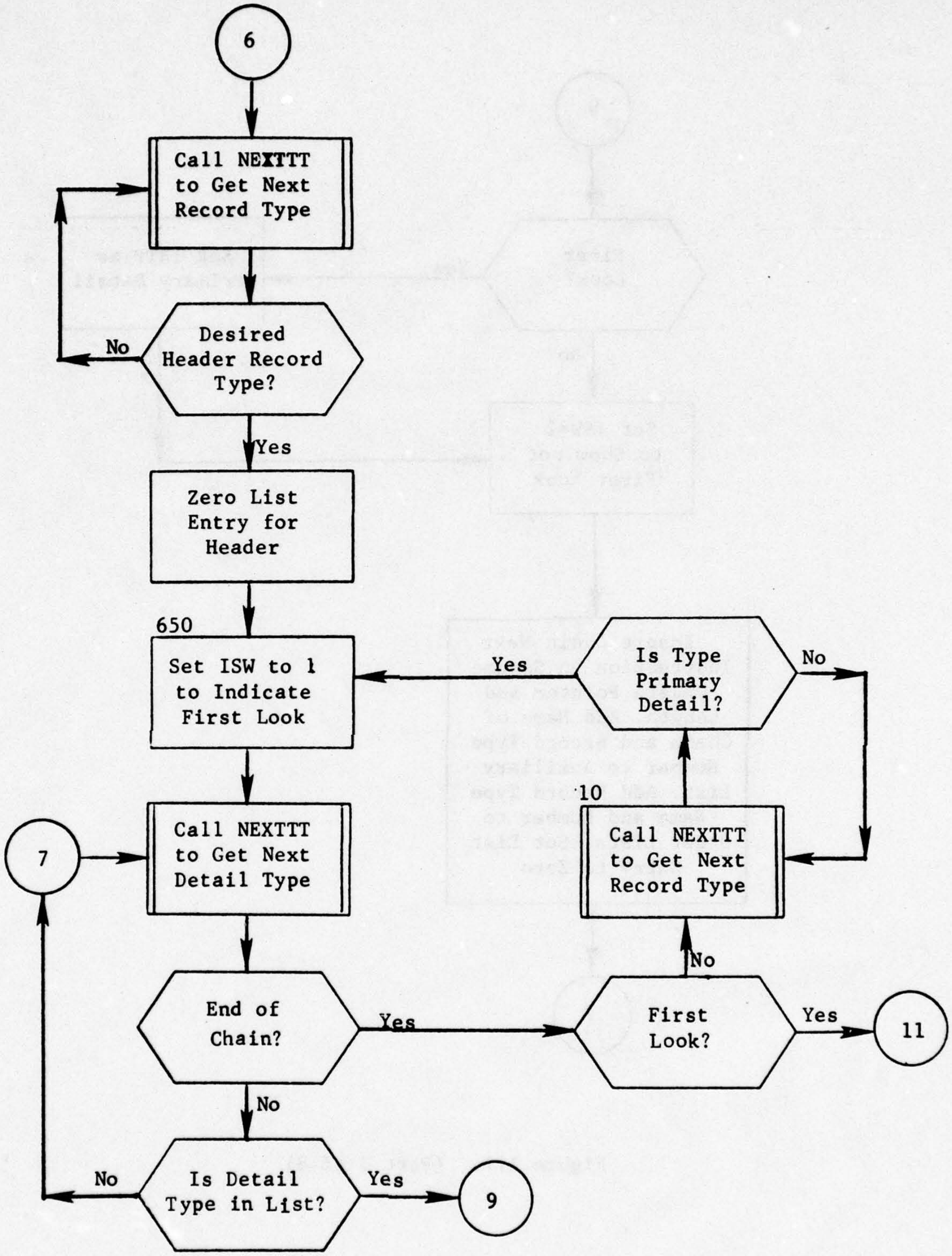


Figure 177. (Part 2 of 8)

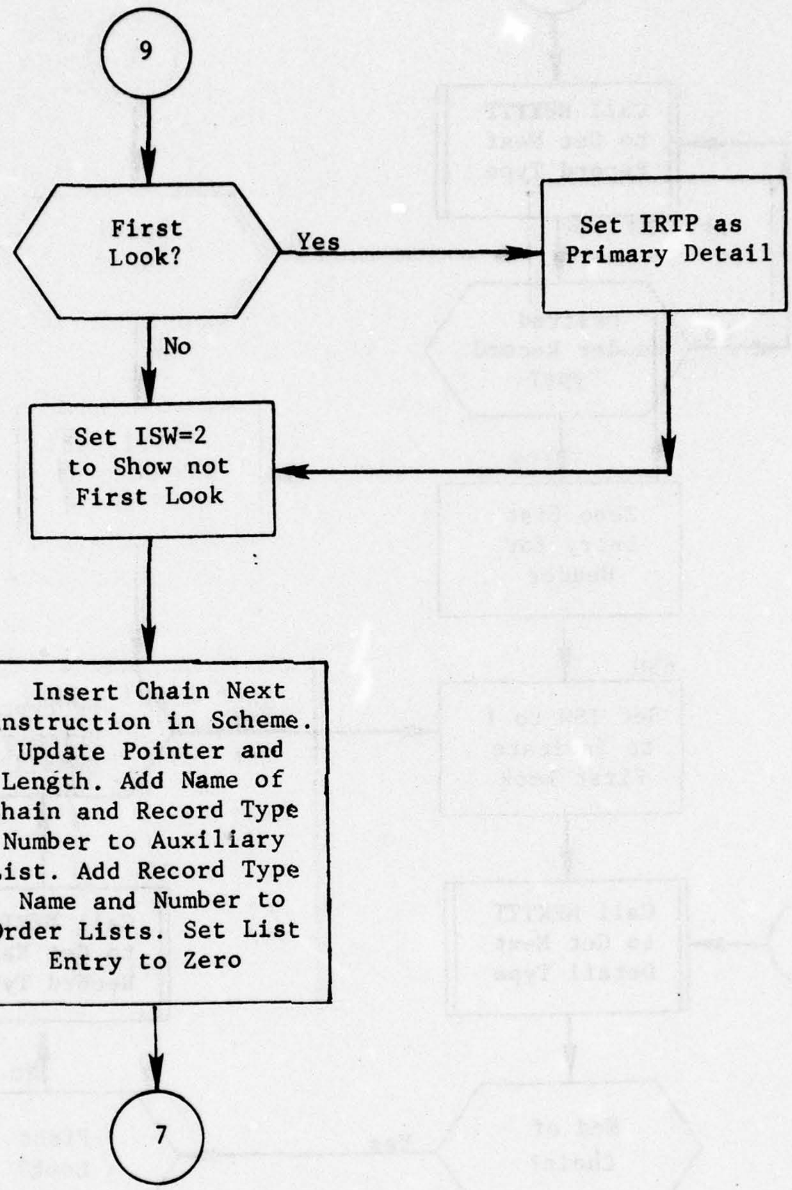


Figure 177. (Part 3 of 8)

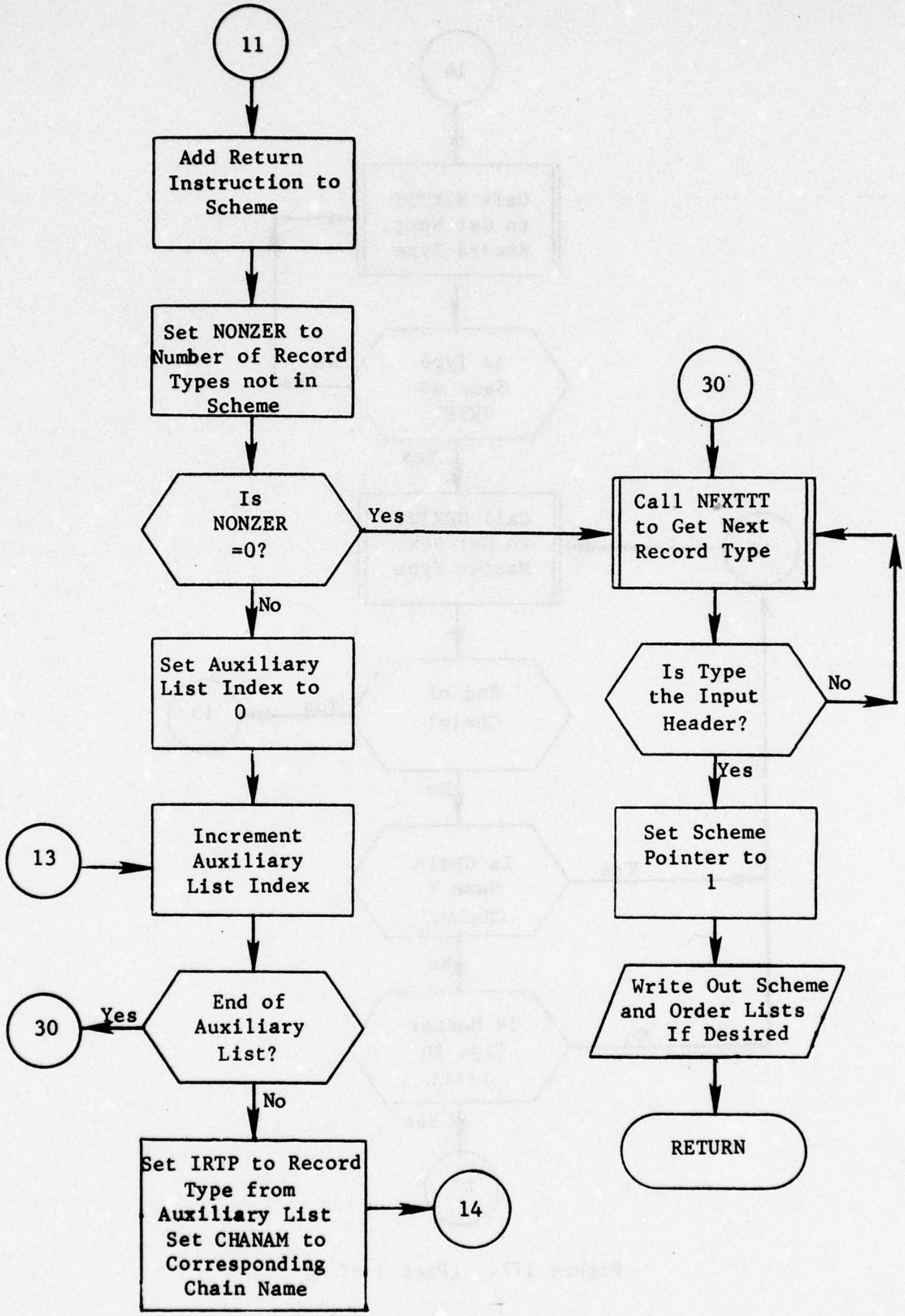


Figure 177. (Part 4 of 8)

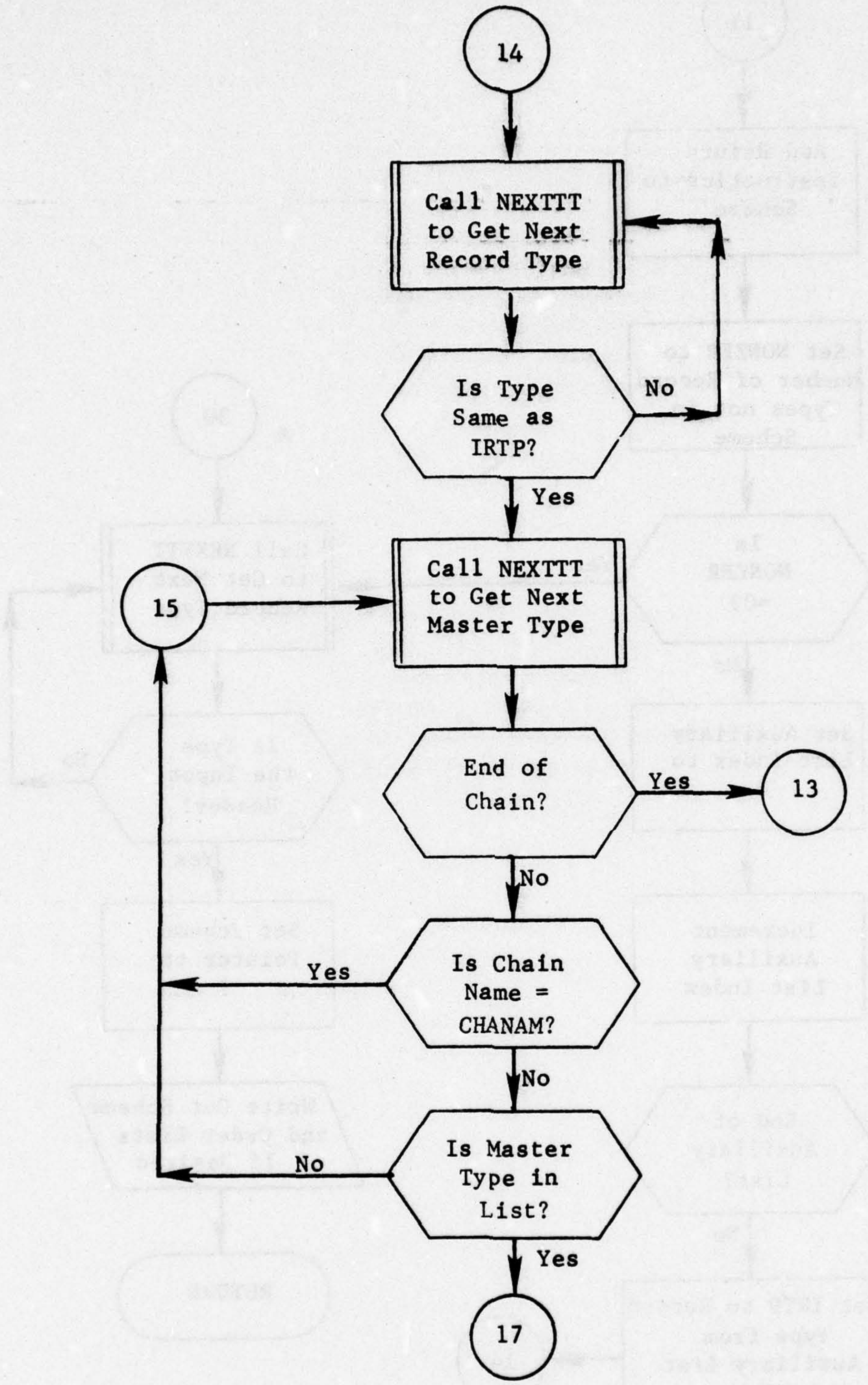


Figure 177. (Part 5 of 8)

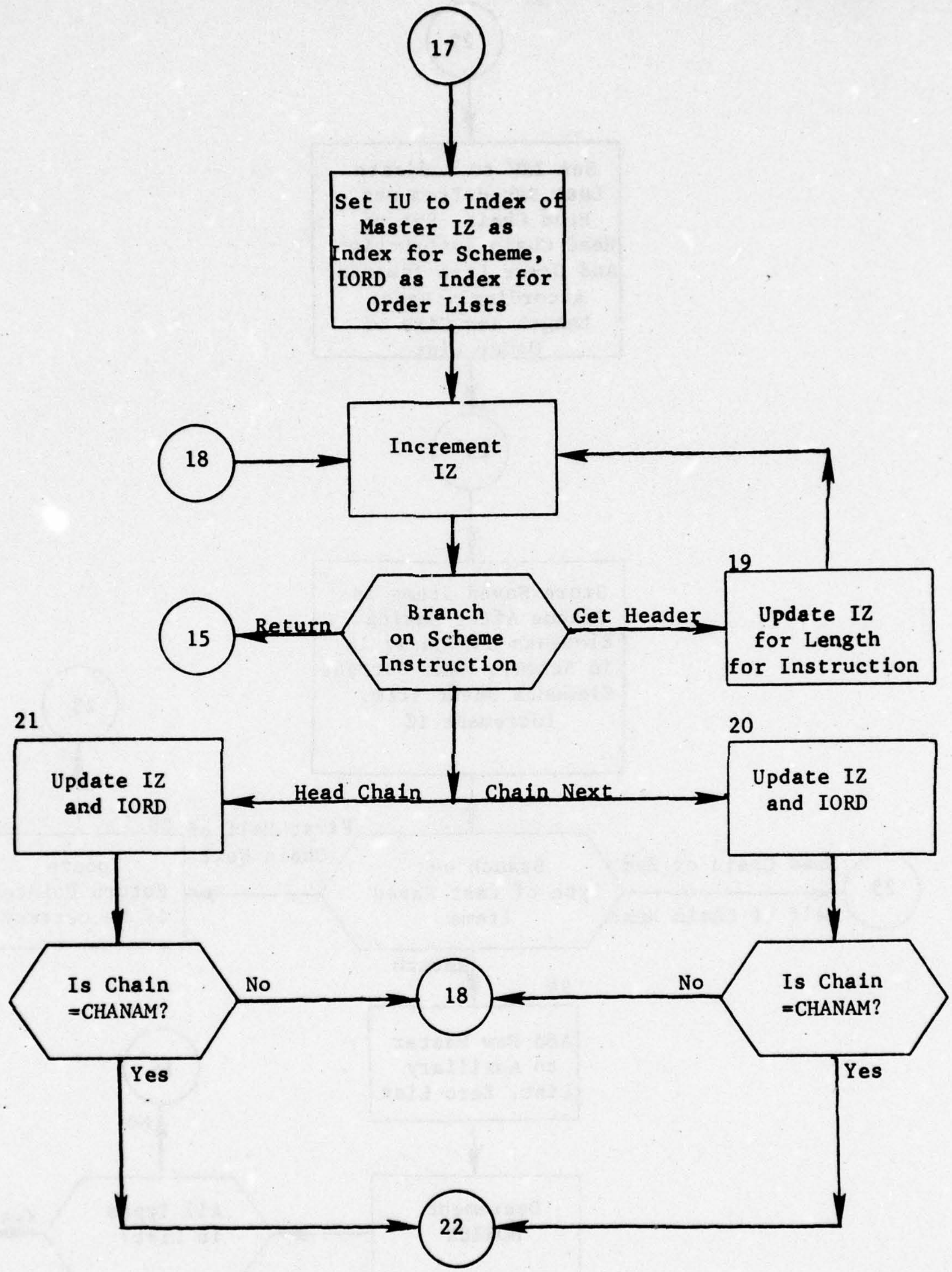


Figure 177. (Part 6 of 8)

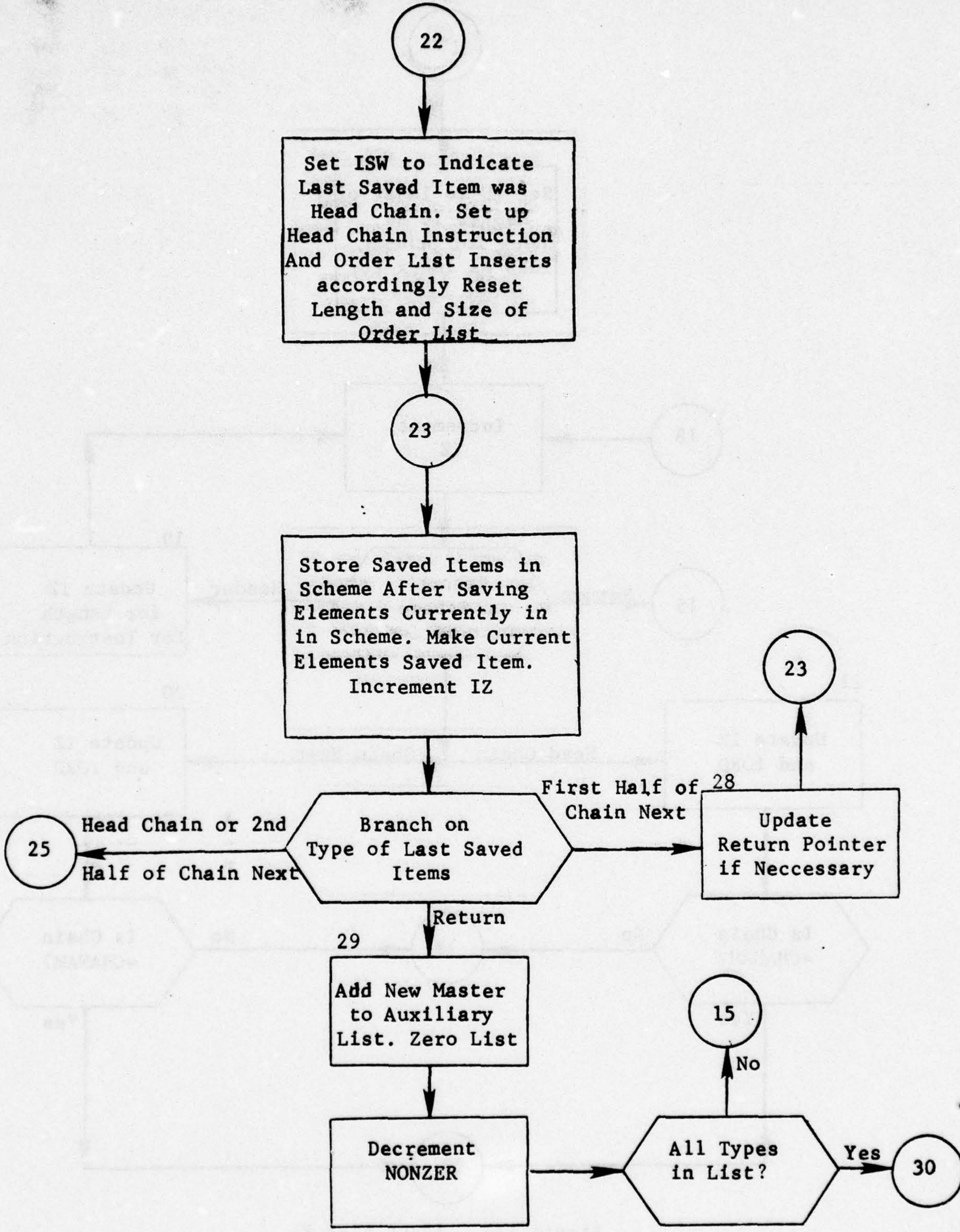


Figure 177. (Part 7 of 8)

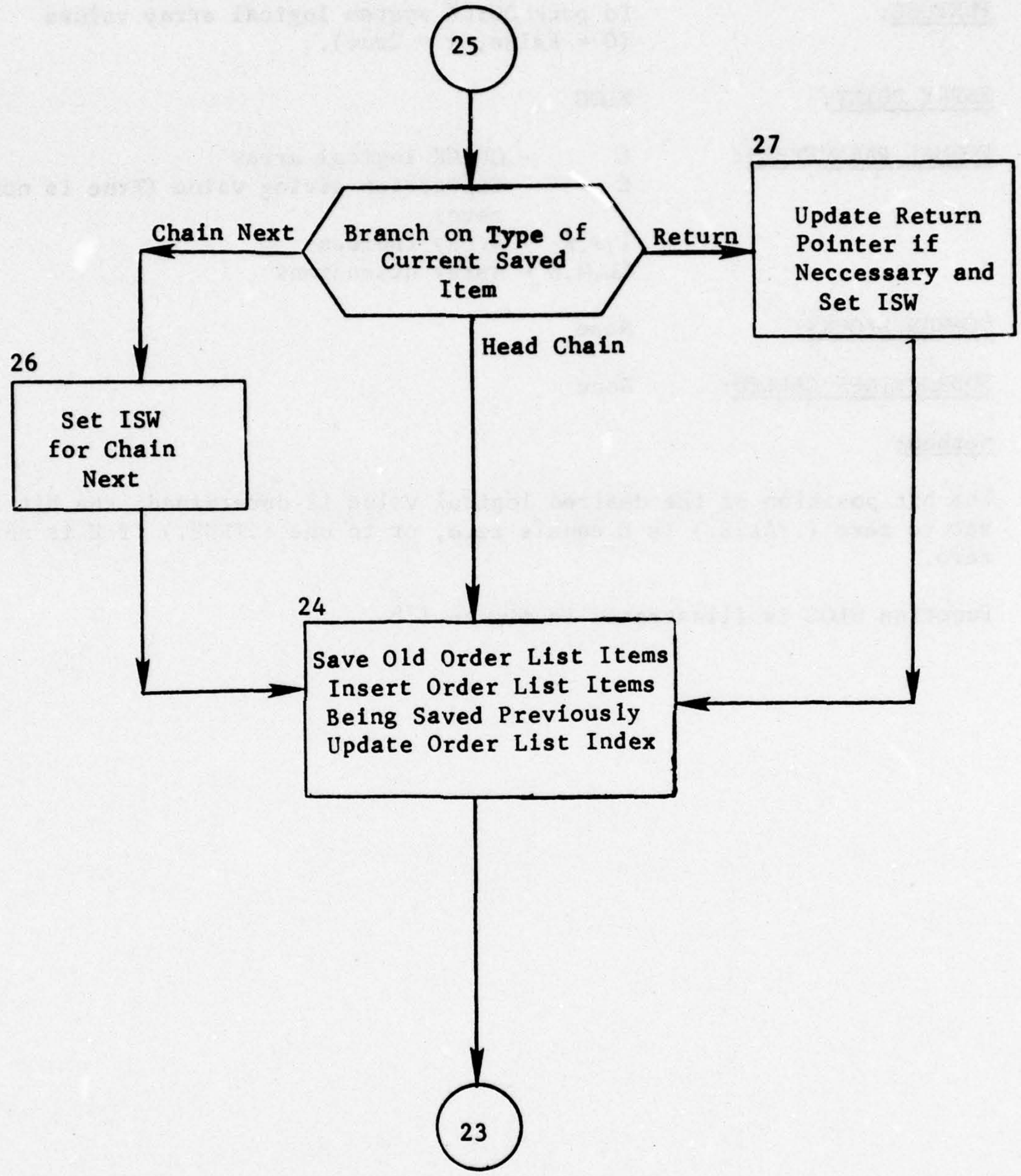


Figure 177. (Part 8 of 8)

9.48 Function SLOG

PURPOSE: To pack QUICK system logical array values (0 = False; 1 = True).

ENTRY POINT: SLOG

FORMAL PARAMETERS:

L	- QUICK logical array
E	- Expression giving value (True is non-zero)
I,J,K	- Array indices
LL,M,N	- Array dimensions

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method:

The bit position of the desired logical value is determined; the bit is set to zero (.FALSE.) is E equals zero, or to one (.TRUE.) if E is non-zero.

Function SLOG is illustrated in figure 178.

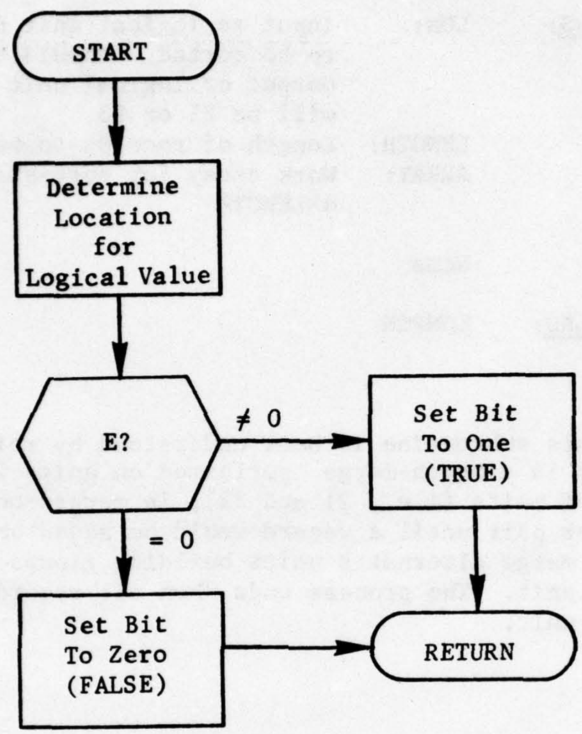


Figure 178. Function SLOG

9.49 Subroutine SORTIT

PURPOSE: Generalized external four unit sort

ENTRY POINTS: SORTIT

FORMAL PARAMETERS:

- LUN: Input as logical unit numbers of file to be sorted - should be 21. Output or logical unit of sorted file - will be 21 or 23
- LENGTH: Length of records to be sorted in words
- ARRAY: Work array for sort-size should be 4*LENGTH

COMMON BLOCKS: None

SUBROUTINES CALLED: KOMPCH

Method:

The method of this subroutine is best understood by reference to figure 179. The method is a match-merge performed on units 21, 22, 23, and 24. Each pair of units (i.e., 21 and 22), is merged onto the first unit of the other pair until a record would be added which is out of sort. Then the merge alternates units building groups of in-sort records on each unit. The process ends when all records have been merged onto one unit.

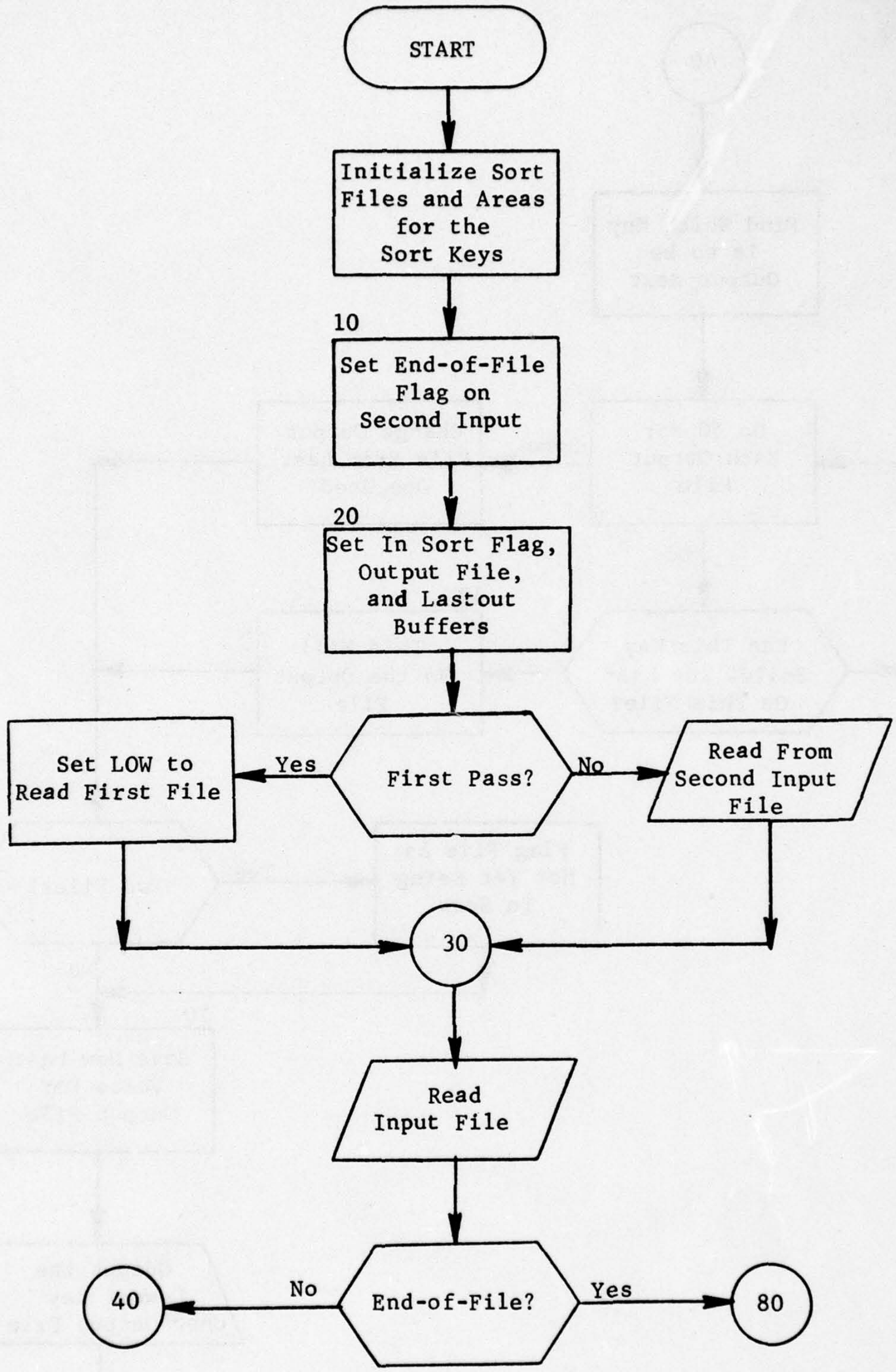


Figure 179. Subroutine SORTIT (Part 1 of 3)

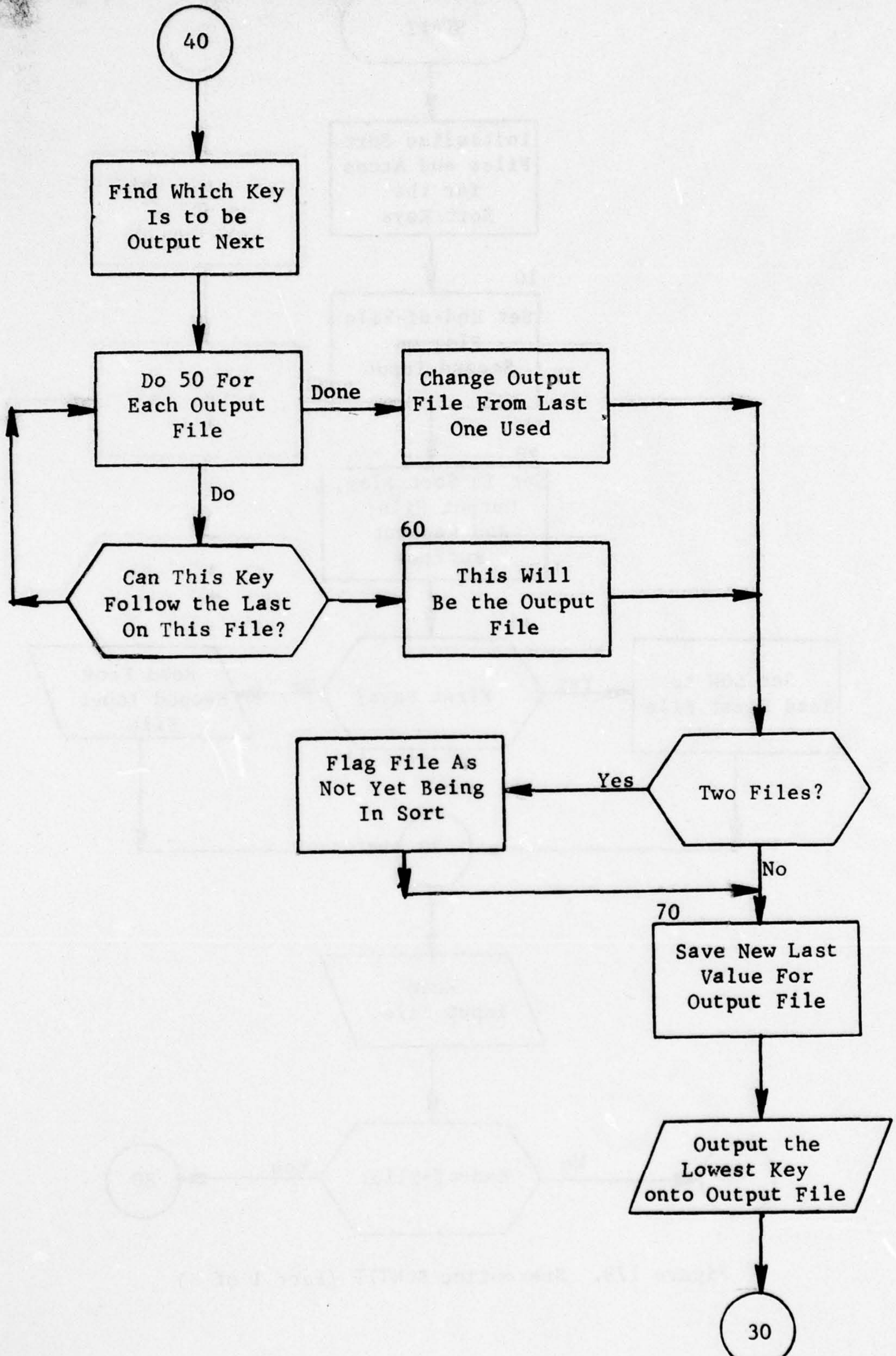


Figure 179. (Part 2 of 3)

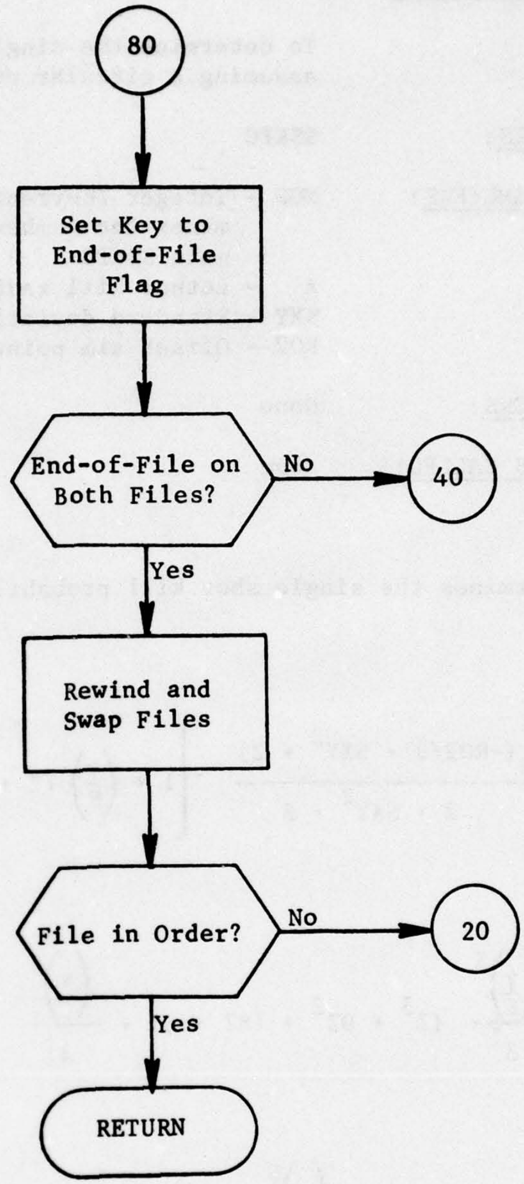


Figure 179. (Part 3 of 3)

9.50 Function SSKPC

PURPOSE: To determine the single shot kill probability assuming a circular normal delivery distribution.

ENTRY POINTS: SSKPC

FORMAL PARAMETERS:

- MOD - Integer (currently 1, 3, or 6); MOD determines the number of terms used to approximate SSKPC
- A - Lethal kill radius (positive real number)
- SXY - Standard deviation
- RO2 - Offset aim point distance, squared

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method:

SSKPC determines the single shot kill probability PKW using the following formula:

$$PKW = \frac{e^{(-RO2/2 \cdot SXY^2 + Z)}}{2 \cdot SXY^2 \cdot \beta} \Bigg[1 + \left(\frac{\gamma}{\beta}\right)(Z + 1) + \frac{\left(\frac{\gamma}{\beta}\right)^2}{2!}(Z^2 + 4Z + 2)$$

$$+ \frac{\left(\frac{\gamma}{\beta}\right)^3}{3!}(Z^3 + 9Z^2 + 18Z + 6) + \frac{\left(\frac{\gamma}{\beta}\right)^4}{4!}(Z^4 + 16Z^3 + 72Z^2$$

$$+ 96Z + 24) + \frac{\left(\frac{\gamma}{\beta}\right)^5}{5!}(Z^5 + 25Z^4 + 200Z^3 + 600Z^2 + 600Z + 120) \Bigg]$$

where $$\gamma = \frac{MOD}{\Lambda^2}, \quad \beta = \frac{MOD}{\Lambda^2} + \frac{1}{2 \cdot SXY^2}$$

and $$Z = \frac{RO2}{4 \cdot \beta \cdot SXY^4}$$

The flow diagram in figure 180 indicates the way in which this formula is used to calculate PKW for different values of MOD (MOD is the equivalent of the parameter W as described under <u>Derivation of Kill Probability Function</u>, Analytical Manual, Volume II). Note that BETA = β, GAM = γ, GB = GAM/BETA = γ/β, and

$$C = \frac{e^{(-RO2/2 \cdot SXY^2 + Z)}}{2 \cdot SXY^2 \cdot BETA}$$

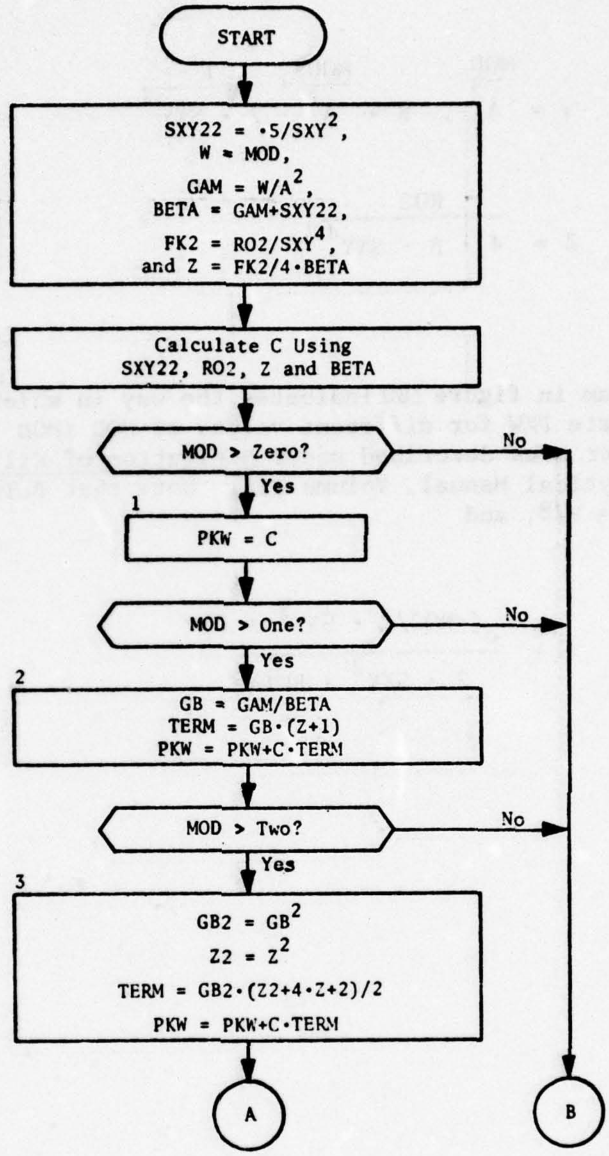


Figure 180. Function SSKPC (Part 1 of 2)

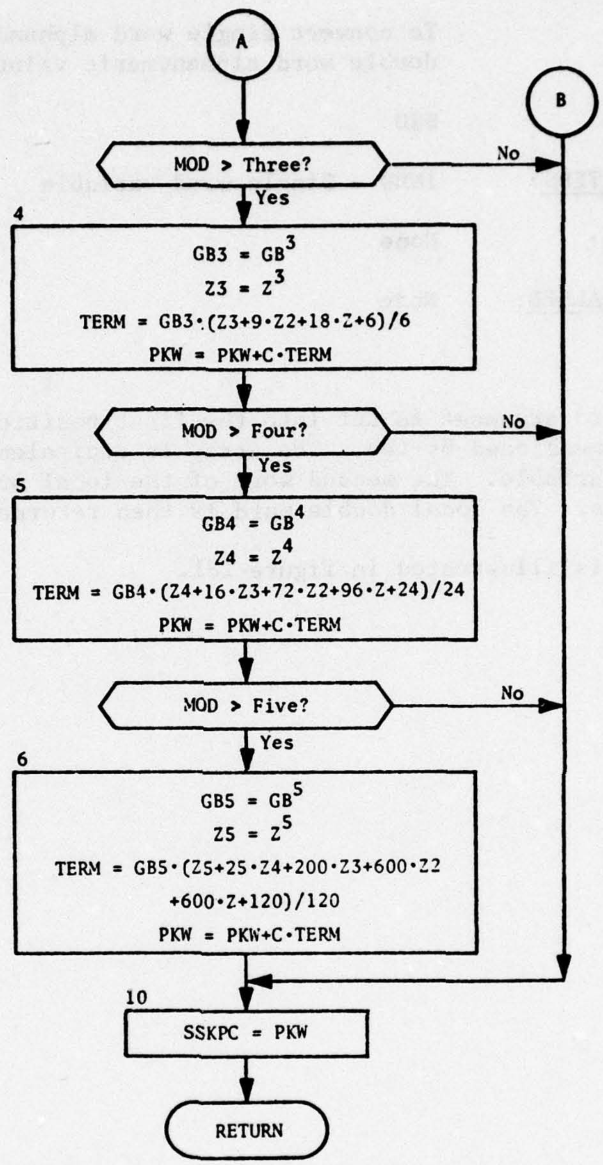


Figure 180. (Part 2 of 2)

9.51 Function STD

PURPOSE: To convert single word alphanumeric values to double word alphanumeric values (CHARACTER*8).

ENTRY POINTS: STD

FORMAL PARAMETERS: IWRD - Single word variable

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method:

The single word argument is set into the first position of a local single word array dimensioned by two. The array is equivalenced to a local double word variable. The second word of the local array is data set to contain blanks. The local double word is then returned in STD.

Function STD is illustrated in figure 181.

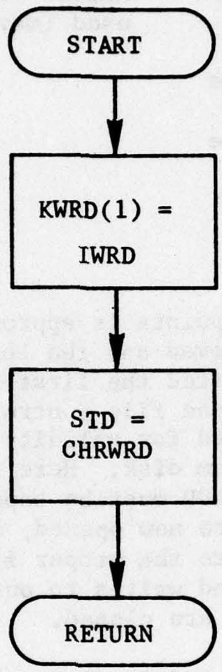


Figure 181. Function STD

9.52 Subroutine SVTP

PURPOSE:	To save a random disk file on tape or load one from tape
ENTRY POINTS:	FILLOD, FILSAV
FORMAL PARAMETERS:	FROM: File code of input TO: File code of output FILE: Number of file on multifile tape to be used (may be omitted)
COMMON BLOCKS:	None
SUBROUTINES CALLED:	None
CALLED BY:	SRM

Method:

The method for both entry points is approximately the same. The input and output codes are retrieved and the third parameter (FILE) is checked for. If it is omitted the first tape file is used. A 4k buffer is now created and the file control blocks set up. The input and output codes are checked for validity and it is assured that one is tape and the other random disk. Here the entry points differ in that the input unit to FILLOD must be tape and the output from FILSAV must be tape. The files are now opened, depending on the third parameter, the tape is spaced to the proper file. The subroutine now loops through reads from input and writes to output until the input unit is exhausted. Then the files are closed.

Subroutine SVTP is illustrated in figure 182.

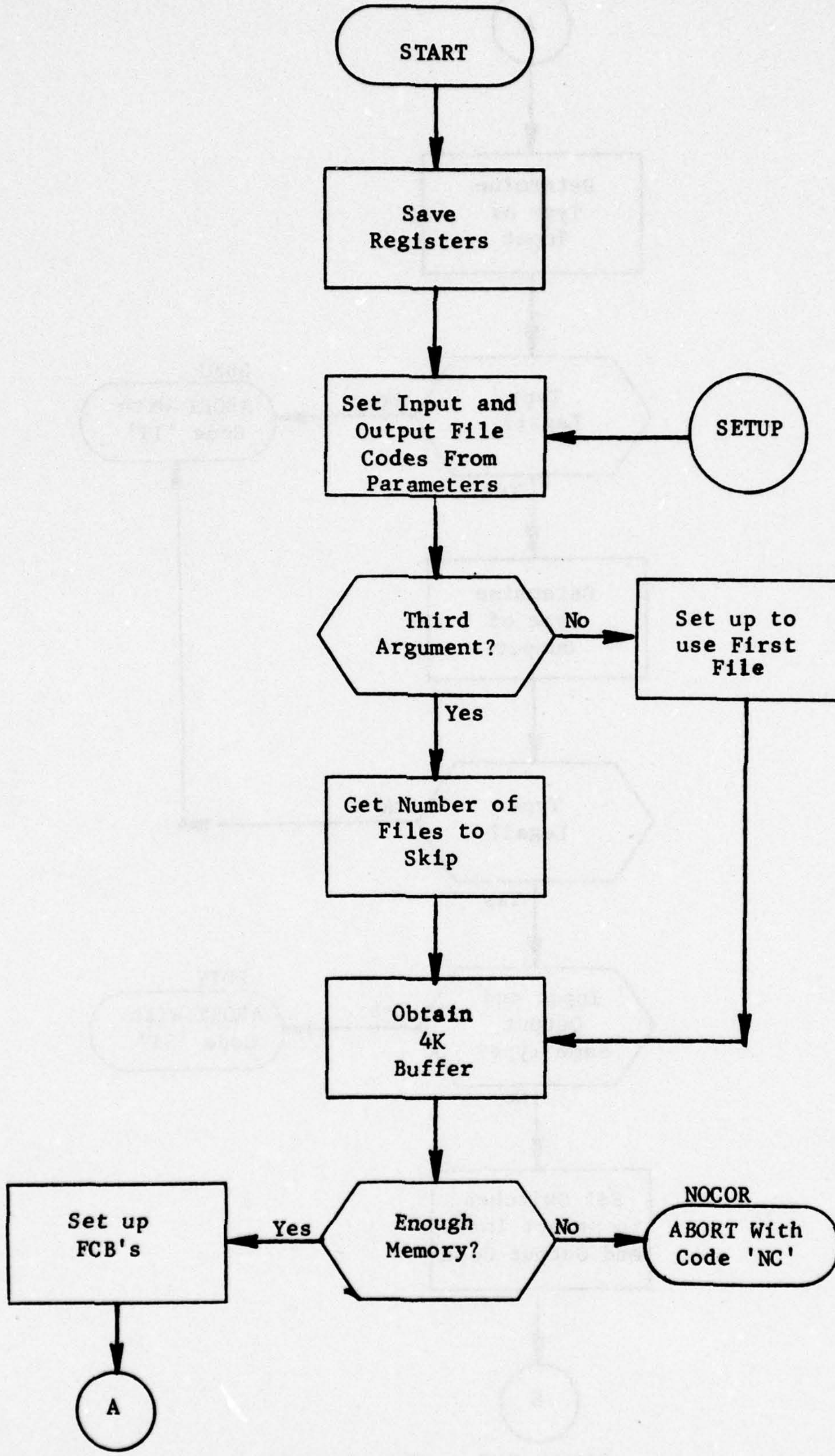


Figure 182. Subroutine SVTP: Entry FILLOD
(Part 1 of 5)

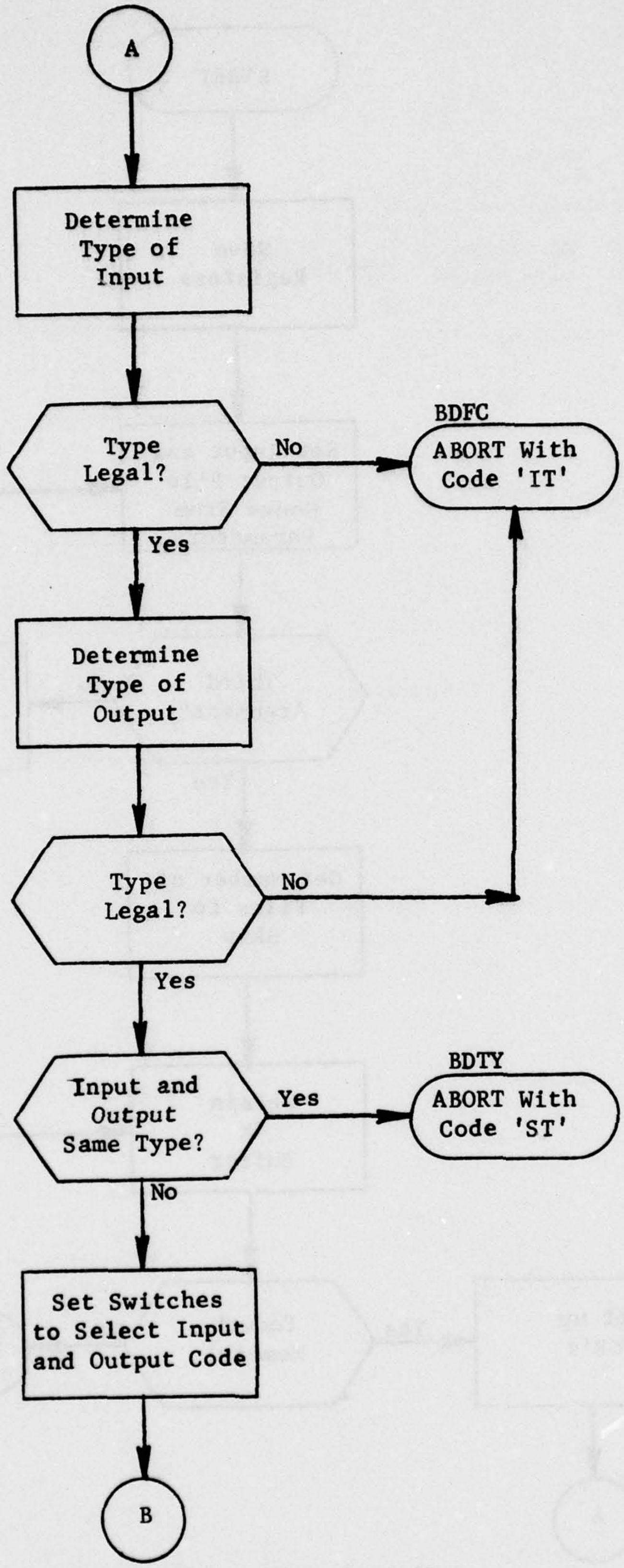


Figure 182. (Part 2 of 5)

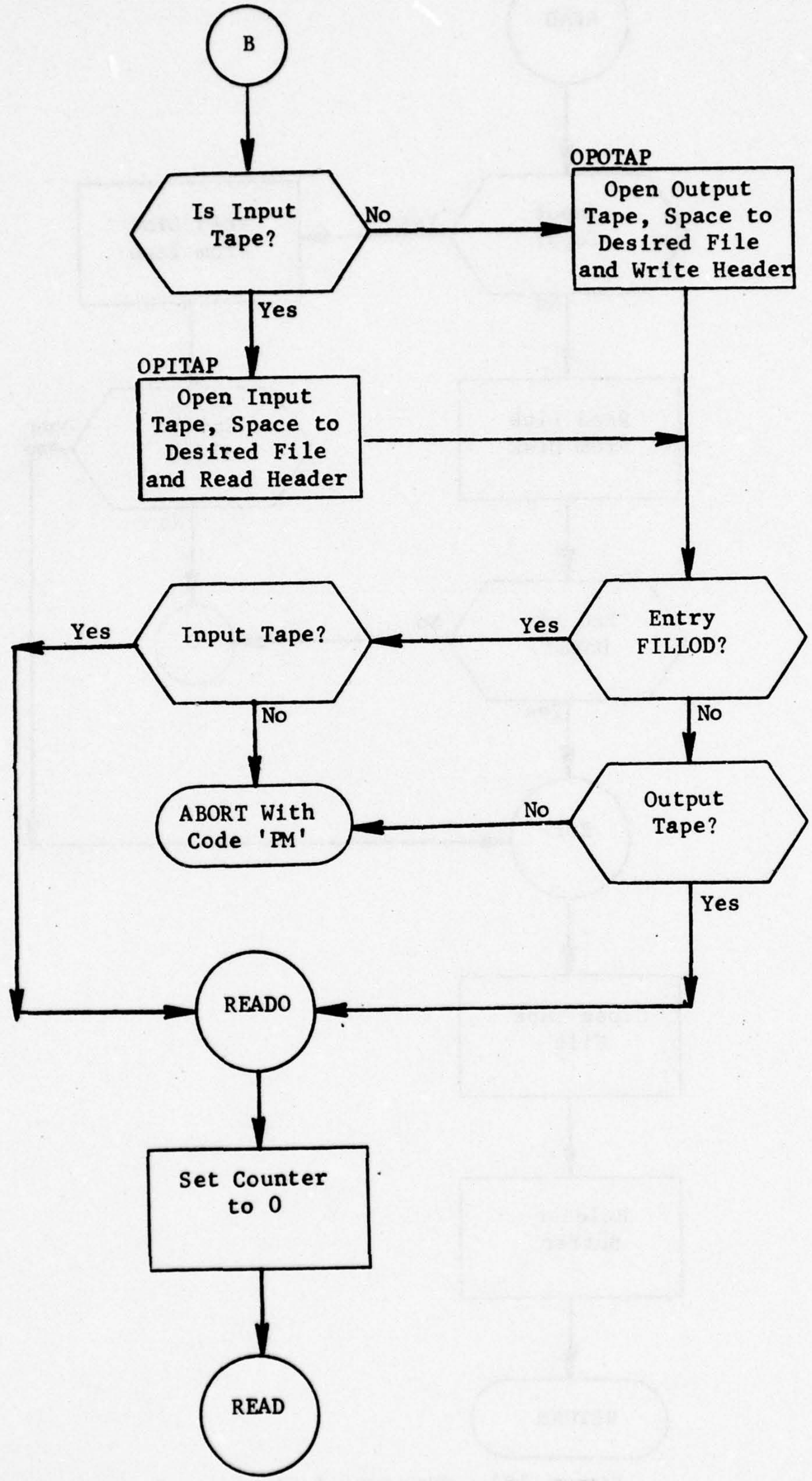


Figure 182. (Part 3 of 5)

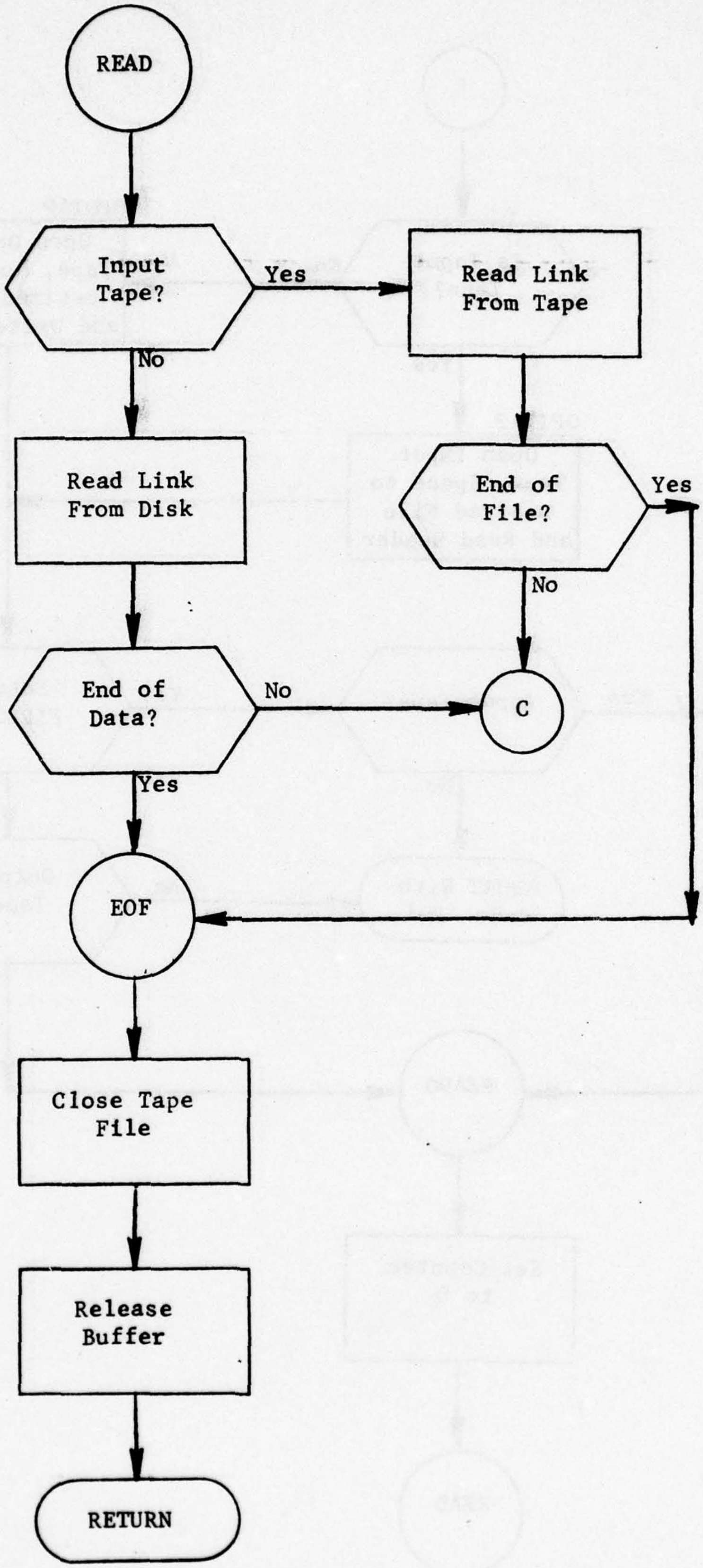


Figure 182. (Part 4 of 5)

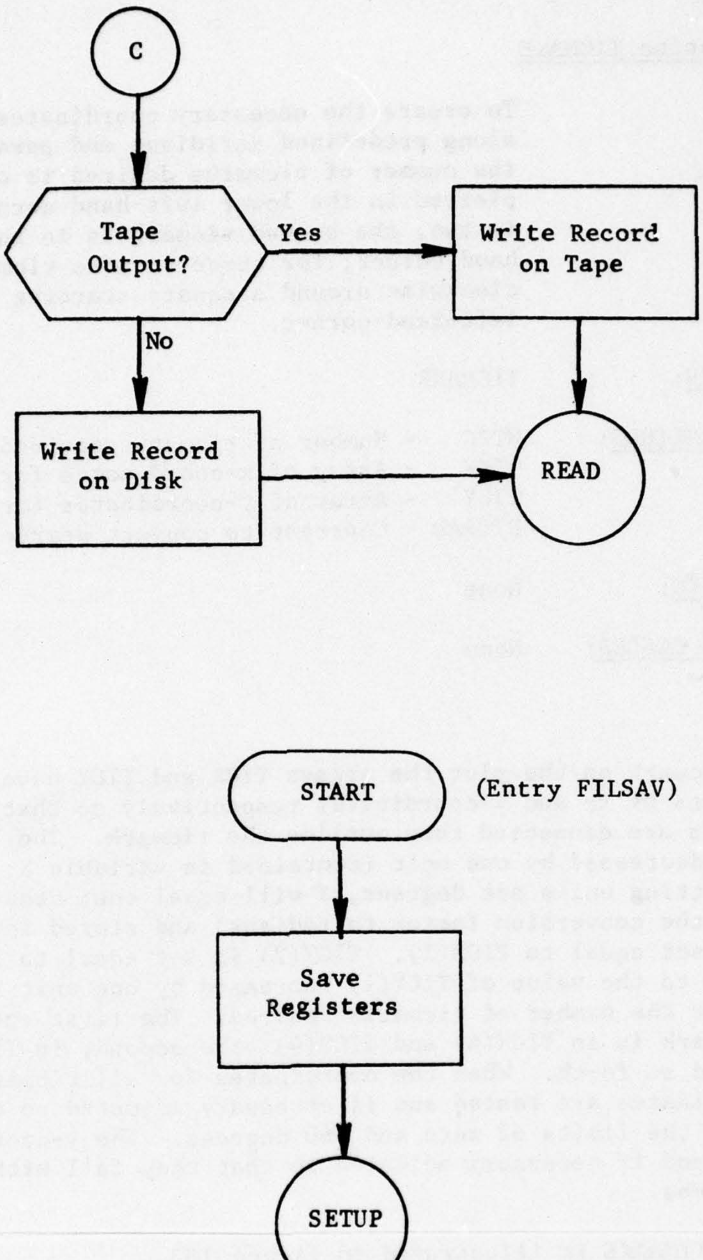


Figure 182. Subroutine SVTP: Entry FILSAV
(Part 5 of 5)

9.53 Subroutine TICMAKE

PURPOSE: To create the necessary coordinates for ticmarks along predefined meridians and parallels. If the number of ticmarks desired is one, it is plotted in the lower left-hand corner. If it is two, the second ticmark is in the upper right-hand corner; for three or more ticmarks they go clockwise around a square starting in the lower left-hand corner.

ENTRY POINTS: TICMAKE

FORMAL PARAMETERS:

NTIC - Number of ticmarks desired
TICX - Array of x-coordinates for ticmarks
TICY - Array of y-coordinates for ticmarks
DTORAD - Constant to convert degrees to radians

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method:

For each ticmark on the plot the arrays TICX and TICY have to contain three entries of x- and y-coordinates respectively so that when the three points are connected they outline the ticmark. The value of TICX(1) is decreased by one unit (contained in variable X: if the desired plotting units are degrees, X will equal one; otherwise, X will equal the conversion factor to radians) and stored in TICX(2). TICX(3) is set equal to TICX(1). TICY(2) is set equal to TICY(1) and TICY(3) to the value of TICY(1) increased by one unit X. This is repeated for the number of ticmarks desired. The first entry of the second ticmark is in TICX(4) and TICY(4); the second, in TICX(5) and TICY(5); and so forth. When the coordinates for all ticmarks are set, the x-coordinates are tested and if necessary adjusted so that they fall within the limits of zero and 360 degrees. The y-coordinates are tested and if necessary adjusted so that they fall within -90 and 90 degrees.

Subroutine TICMAKE is illustrated in figure 183.

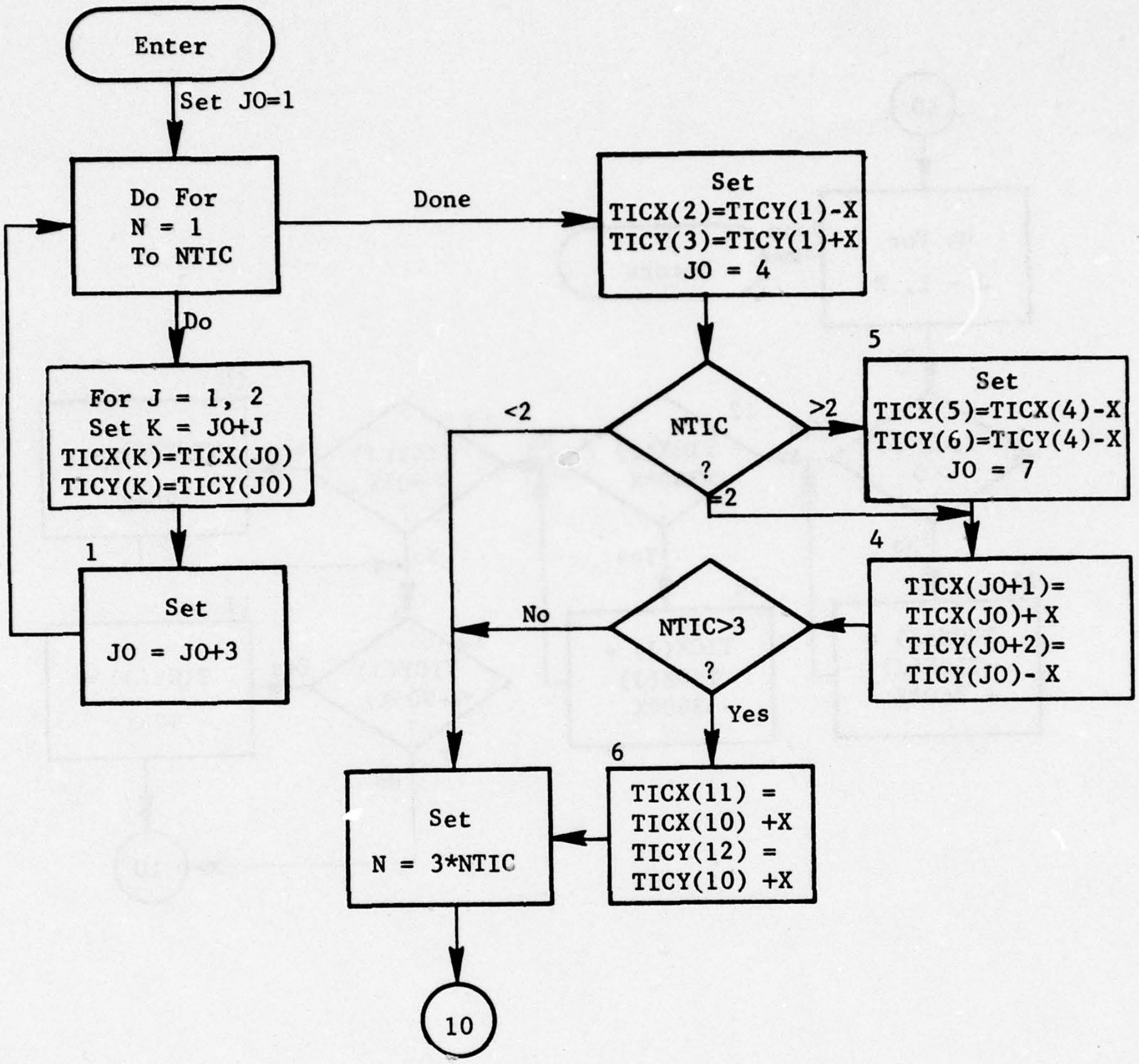


Figure 183. Subroutine TICMAKE (Part 1 of 2)

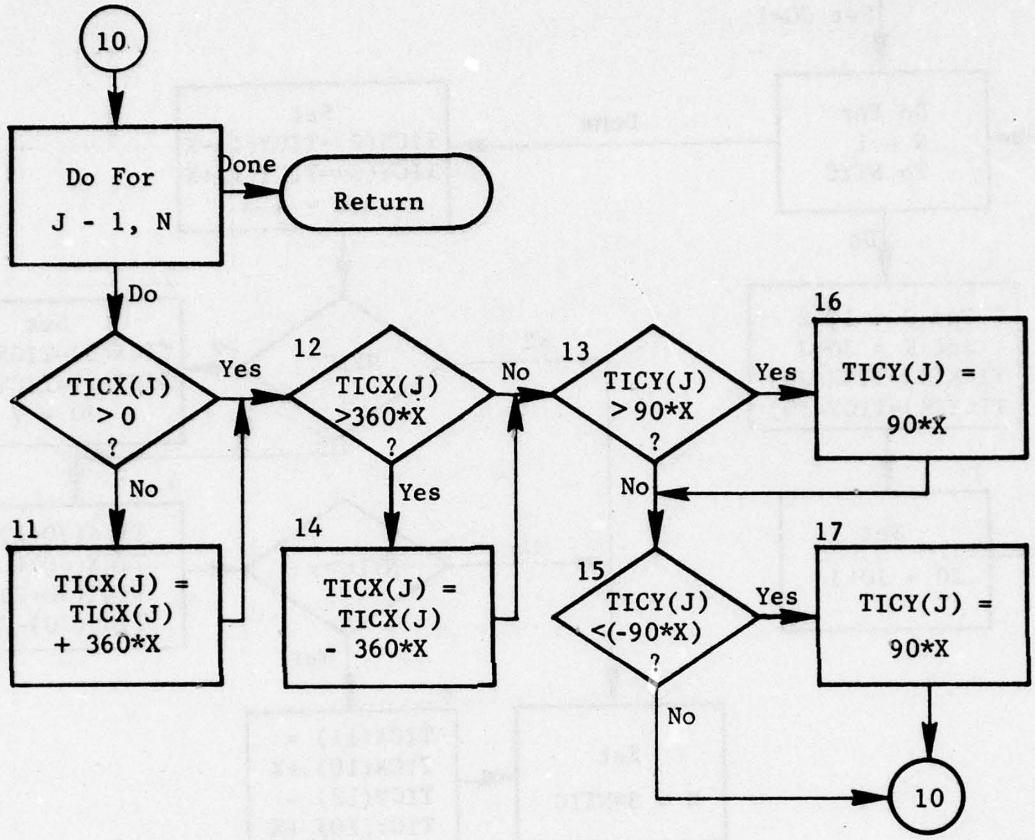


Figure 183. (Part 2 of 2)

9.54 Function TIMEDAY

PURPOSE: To obtain the current time of day.

ENTRY POINTS: TIMEDAY

FORMAL PARAMETERS: DUMMY - A dummy parameter

COMMON BLOCKS: None

SUBROUTINES CALLED: DATIM

Method:

This function is used to place the current time of day in A6 format. The six characters that are returned are used as follows: two characters, hour; two characters, minute; two characters, seconds. The system function DATIM is used to retrieve the time of day in this format. When called from a FORTRAN program, the result is returned in floating point.

Function TIMEDAY is illustrated in figure 184.

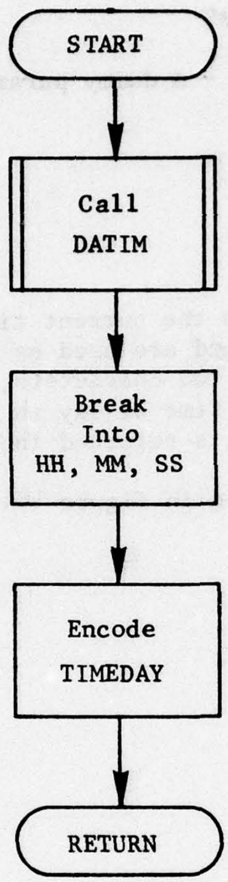


Figure 184. Function TIMEDAY

9.55 Subroutine TIMEME

PURPOSE: To record and print differential and cumulative differential time intervals.

ENTRY POINTS: TIMEME

FORMAL PARAMETERS: I - Integer variable used as described below

COMMON BLOCKS: None

SUBROUTINES CALLED: GETCLOCK

Method:

The formal parameter is used as follows:

I = -3 Reactivate clock

I = -2 Deactivate clock

I = -1 Initialize TIMEME

I = 0 Print differential, cumulative differential, and elapsed times

1≤I≤10 Place differential and cumulative differential times in cell I.

TIMEME is used in QUICK to obtain the differential and cumulative execution times of different stages of a program. A call on TIMEME with I = -1 initializes its internal clock. Subsequent calls with 1≤I≤10 will record the differential time in the Ith differential time cell and will add the differential time into the cumulative differential time cell. For example, to obtain the separate running times for three stages of a program, one would call TIMEME with I = -1 at the beginning of the first stage to initialize the timing. A call with I = 1 at the end of this stage would cause the elapsed time to be recorded in cell 1 of the differential time array and to be added into cell 1 of the cumulative time array. Calls with I = 2 and I = 3 at the end of the second and third stages, respectively, would similarly fill the second and third cells of the two arrays. Notice that I is used as a label for the cells, not as an iteration index.

It is possible to exclude time used for irrelevant operations by using calls with I = -2 and I = -3 at the beginning and end of each irrelevant operation to "turn off" the internal clock. For example, if one has a print subroutine which can be called at any time to produce debug prints, these print times can be excluded from the running times by making a call

with I = -2 at the beginning of the print subroutine and a call with I = -3 at the end. This causes the time elapsed during the print routine to be subtracted from the differential time of the calling stage of the program.

A call with I = 0 causes the differential and cumulative differential time arrays to be printed, as well as the total time, the time lost, and the sum of these two, or the total elapsed time.

Three error warnings may be issued from TIMEME:

a. ILLEGAL TIMEME CALL I = (X)

   (if I > 10)

b. ERROR - STOP ATTEMPTED WITH CLOCK ALREADY STOPPED

   (if two successive calls with I = -2 are made without an intervening call with I = -3)

c. ERROR - RESTART ATTEMPTED WITH CLOCK ACTIVE

   (if two successive calls with I = -3 are made without an intervening call with I = -2)

After each of these messages, the subroutine returns with no further processing.

Subroutine TIMEME is illustrated in figure 185.

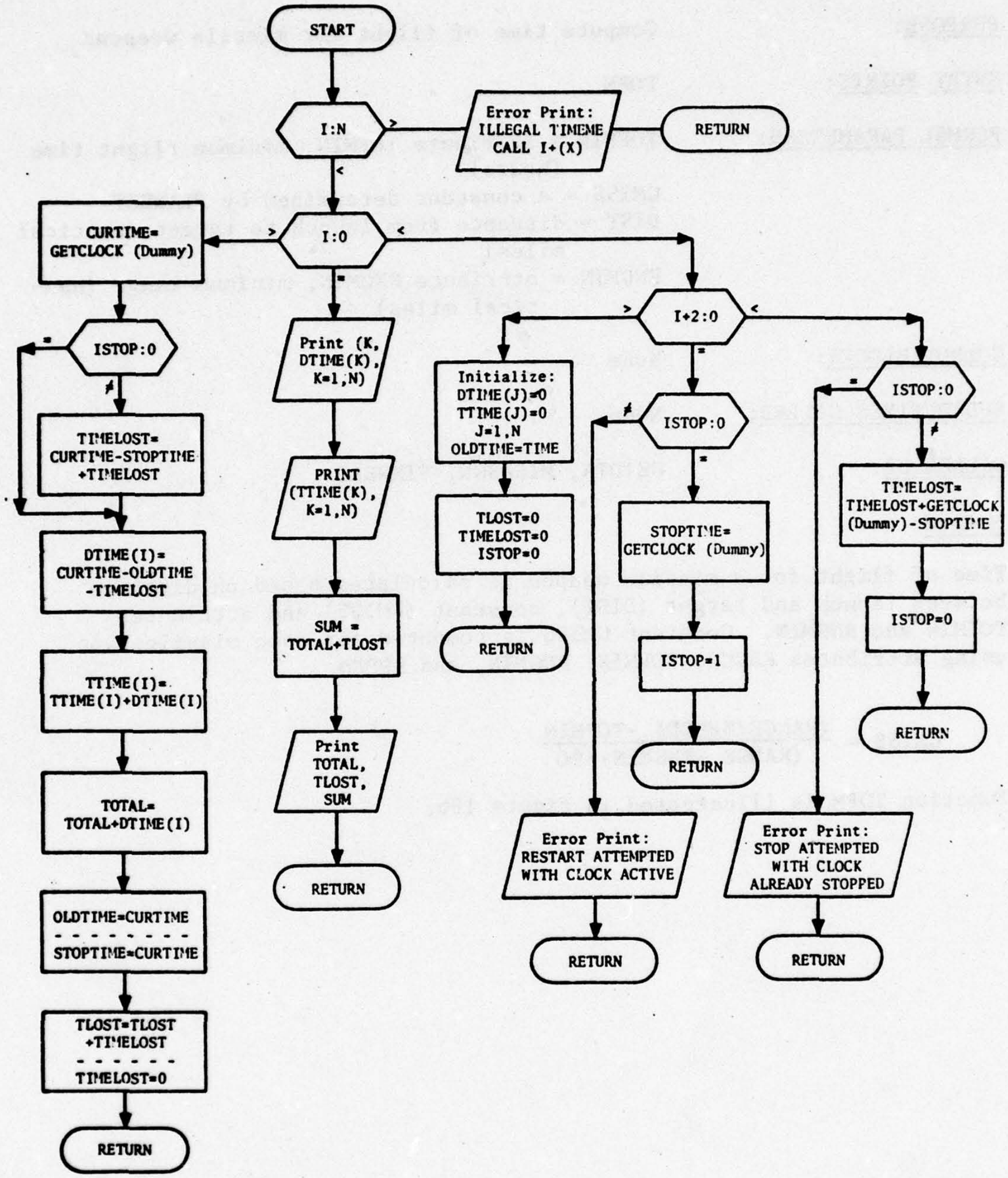


Figure 185. Subroutine TIMEME

9.56 Function TOFM

PURPOSE: Compute time of flight for missile weapons

ENTRY POINTS: TOFM

FORMAL PARAMETERS: TOFMIN = attribute TOFMIN, minimum flight time (hours)
CMISS = a constant determined by PLANSET
DIST = distance from launch to target (nautical miles)
RNGMIN = attribute RNGMIN, minimum range (nautical miles)

COMMON BLOCKS: None

SUBROUTINES CALLED: None

CALLED BY: GETDTA, MISASGN, FIXWEAP

Method:

Time of flight for a missile weapon is calculated based on distance between launch and target (DIST), constant (CMISS), and attributes TOFMIN and RNGMIN. Constant CMISS is computed for each missile type using attributes RANGE, TOFMIN, RNGMIN, and SPEED.

$$CMISS = \frac{(RANGE/SPEED) - TOFMIN}{(RANGE - RNGMIN) \cdot 90}$$

Function TOFM is illustrated in figure 186.

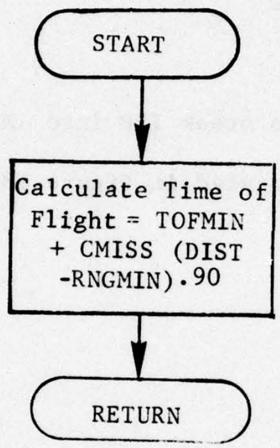


Figure 186. Subroutine TOFM

9.57 Subroutine UNCODE

PURPOSE:	To decode an instruction code
ENTRY POINTS:	UNCODE
FORMAL PARAMETERS:	INP: Input instruction code IW: (bits 29-32) +1 IX: bits 33-35 IY: (bit 33) +1 IZ: bits 34-35
COMMON BLOCKS:	None
SUBROUTINES CALLED:	None

Method:

The method is to use FLD to break INP into IW, IX, IY, and IZ.

Subroutine UNCODE is illustrated in figure 187.

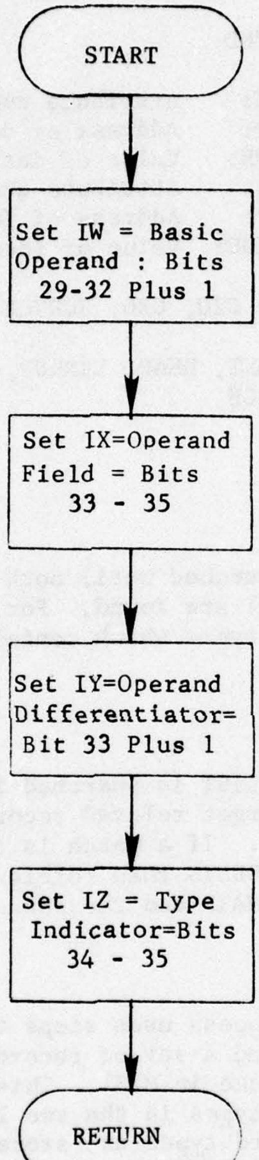


Figure 187. Subroutine UNCODE

9.58 Subroutine VALFND

PURPOSE:	To find the value of an attribute given an identifying attribute's value
ENTRY POINTS:	VALFND
FORMAL PARAMETERS:	IDAN: Attribute number of desired attribute IDAP: Address of desired attribute VALUE: Value of desired attribute (output) JDAN: Attribute number of identifying attribute JDAP: Address of identifying attribute XVALUE: Value of identifying attribute
COMMON BLOCKS:	C10, C20, C30, SCRTCH
SUBROUTINES CALLED:	GETNXT, HEAD, LINKUP, NEXTTT, PRIMHD, RETRV, SETSCH

Method:

Step One

First the ATRIB chain is searched until both identifier attribute (JDAN) and desired attribute (IDAN) are found. For each attribute a list is made of the record type or types which contain these (LIST for IDAN, MIST for JDAN).

Step Two

If JDAN is DESIG (JDAN=3), LIST is searched for a match in the ITGRT array which contains all target related record types. If no match is found an error has occurred. If a match is found, VALFND branches to retrieve the target by its DESIG than retrieve the desired record type. Finally, VALUE is set from MAIN and the subroutine exits.

Step Three

If JDAN is not DESIG the process uses steps three and four. In step three VALFND attempts to find a set of record types which will connect one record type in LIST to one in MIST. This is done by searching the chains of which the record types in the two lists are masters and details. Potential intermediate record types are stored in KIST. The final set of record types is stored in NIST.

Step Four

The primary header (see section 4.4) is now determined by finding the lowest numbered record type and calling PRIMHD. LINKUP and SETSCH are

called to build a retrieval scheme and GETNXT is executed until MAIN (JDAP)=XALUE. Then VALUE is set to MAIN(IDAP) and VALFND exits.

Subroutine VALFND is illustrated in figure 188.

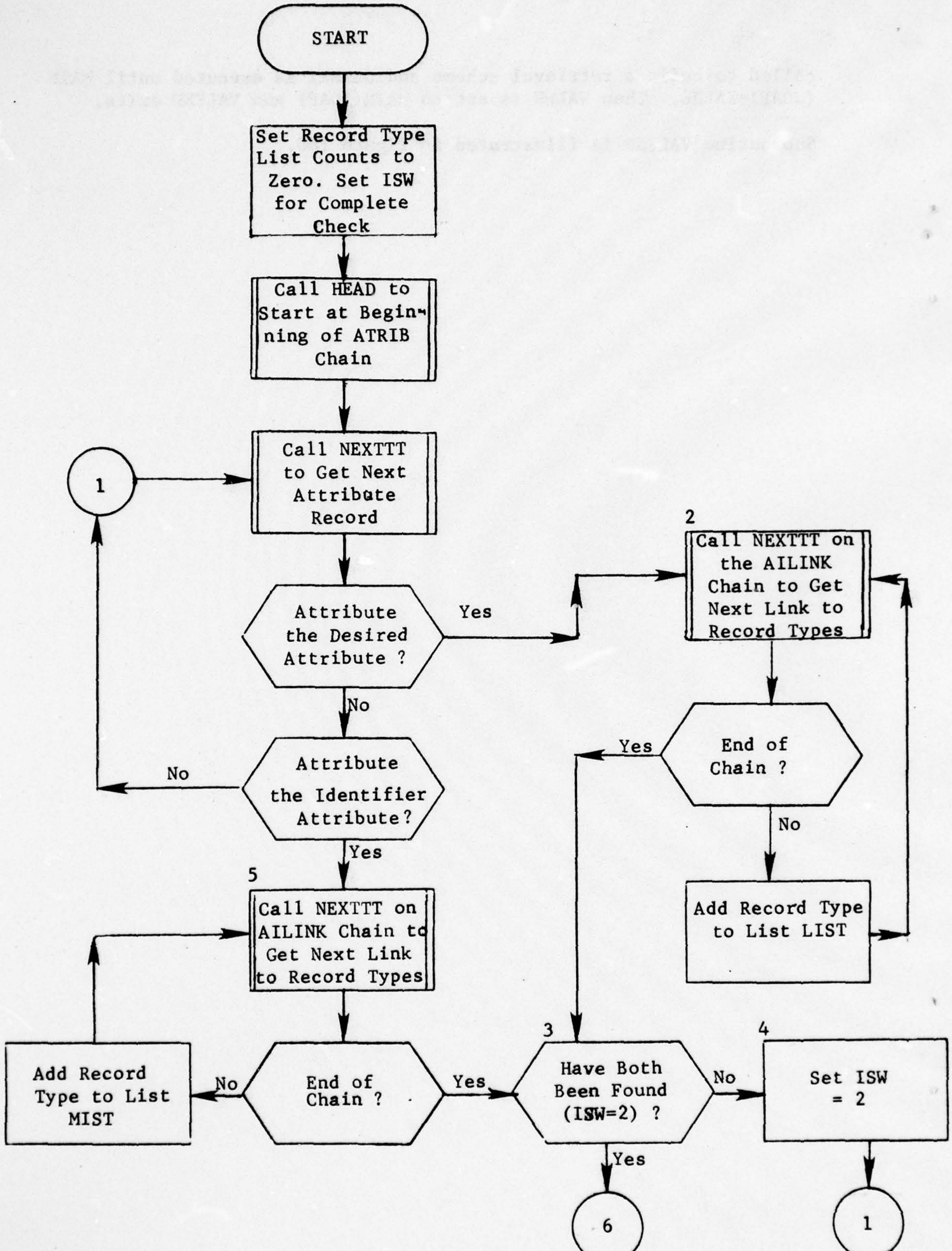


Figure 188. Subroutine VALFND (Part 1 of 8)

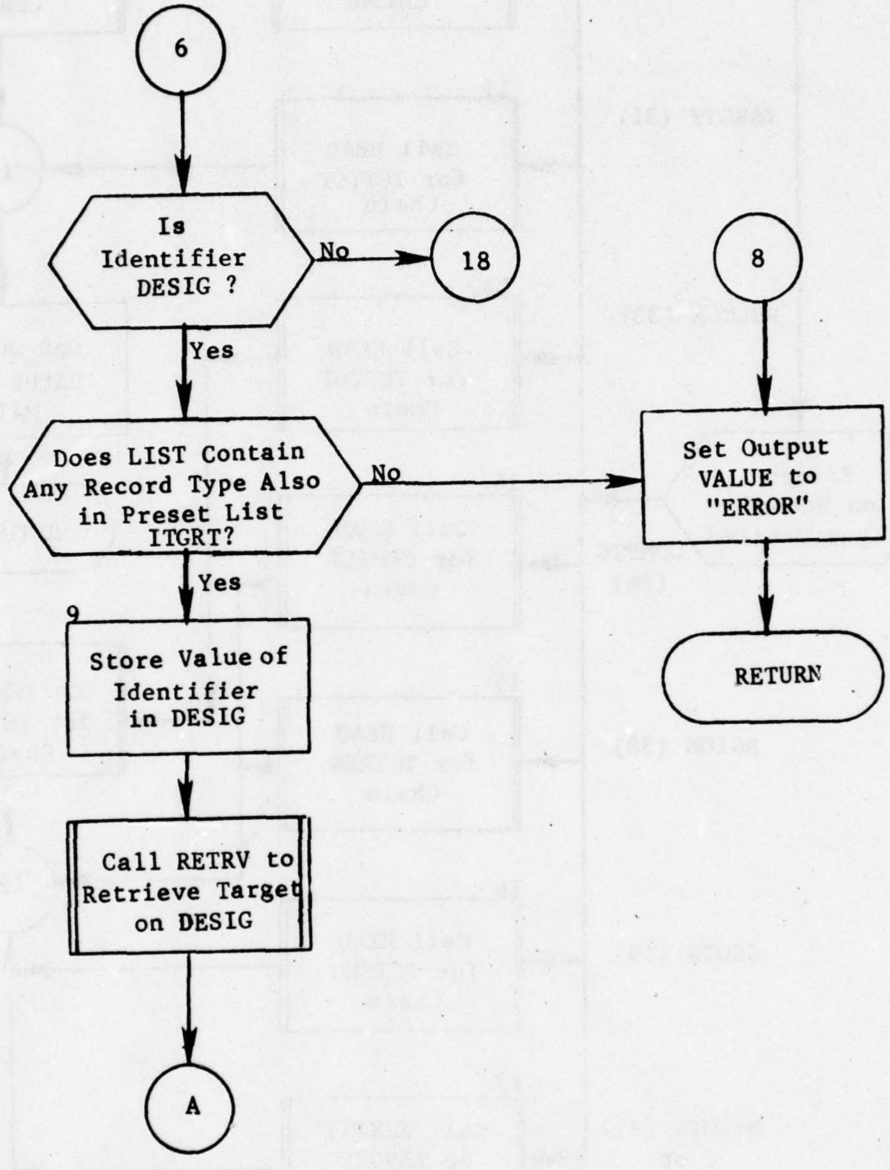
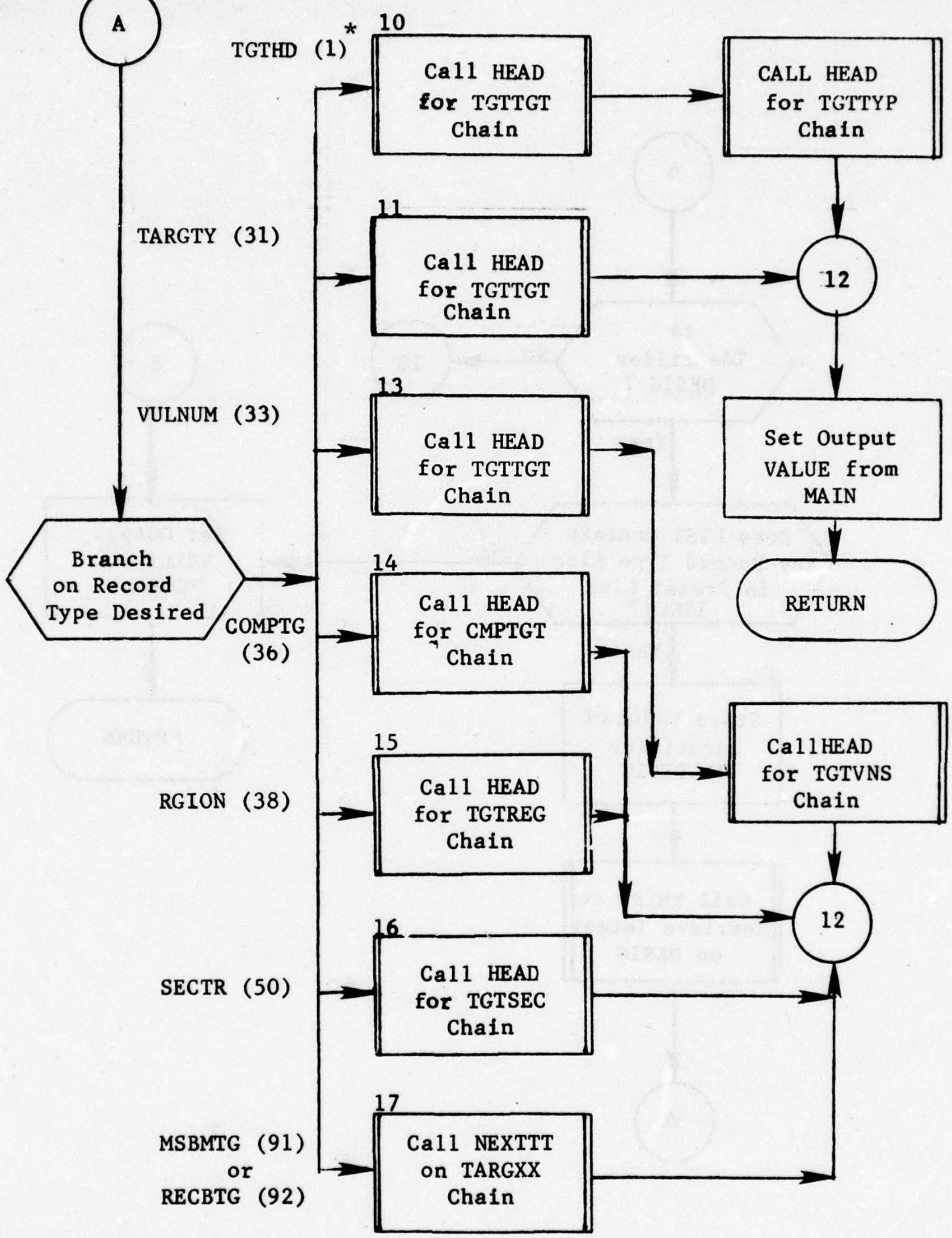


Figure 188. (Part 2 of 8)



* Number in parentheses is Record Type Number preceded by Record Type Name.

Figure 188. (Part 3 of 8)

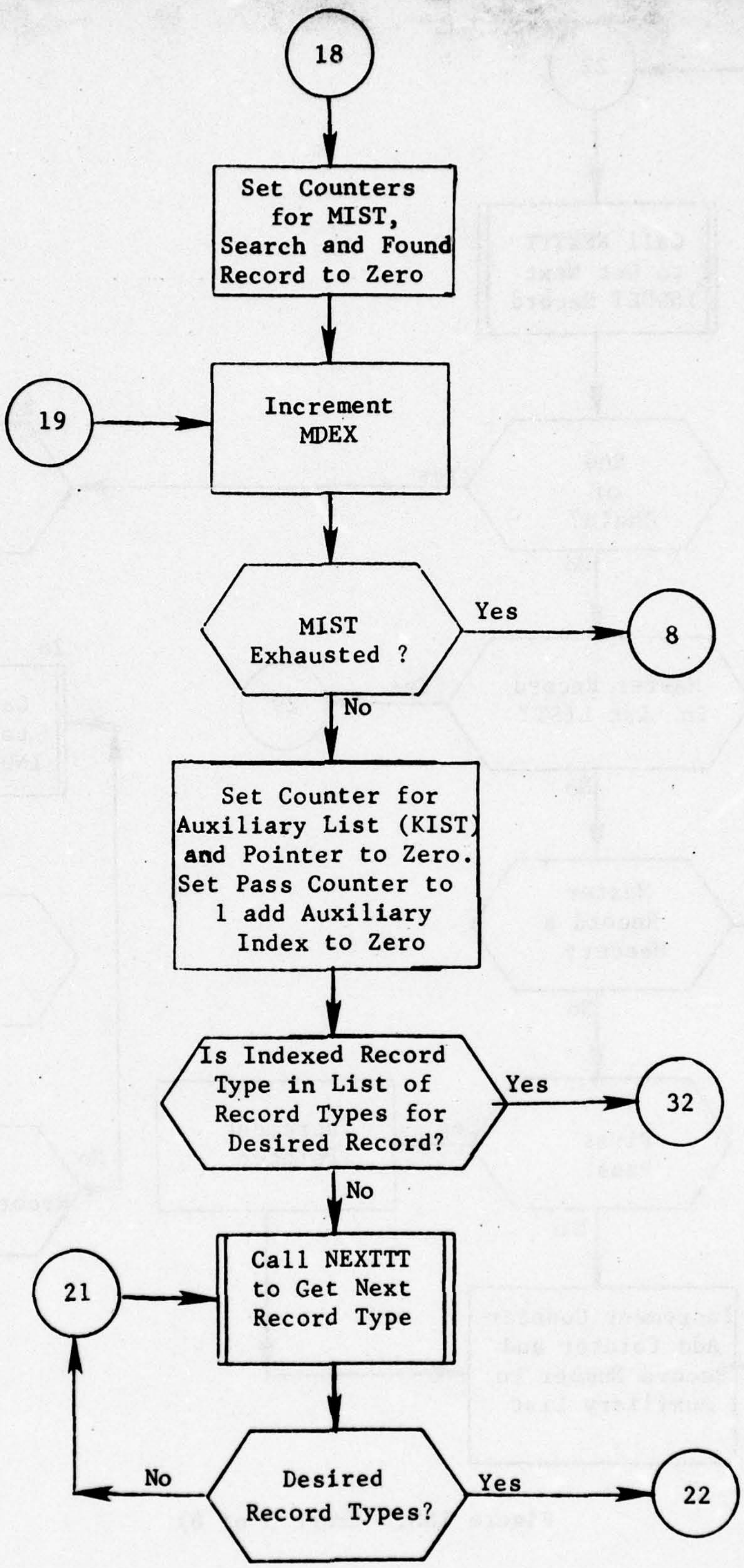


Figure 188. (Part 4 of 8)

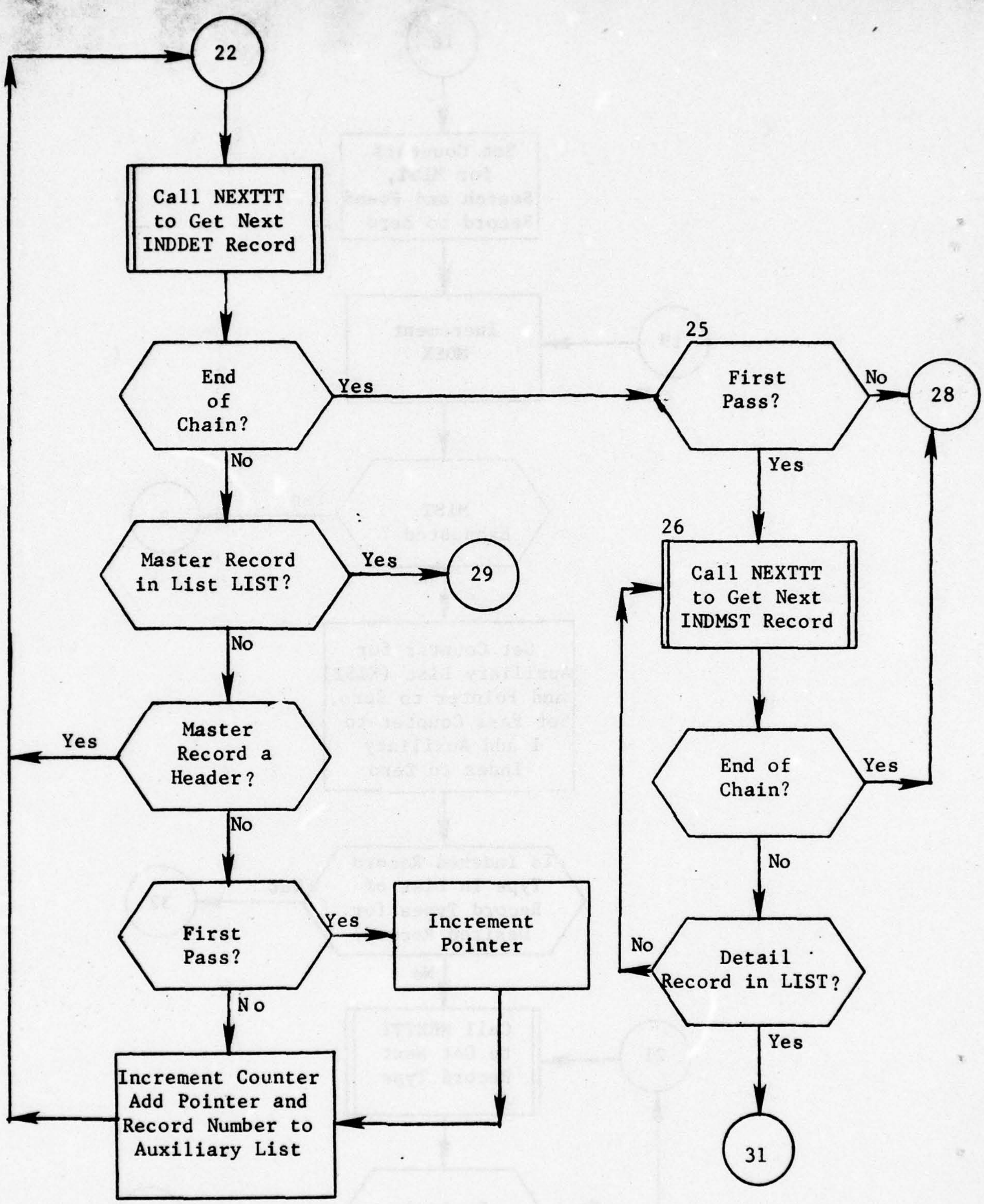


Figure 188. (Part 5 of 8)

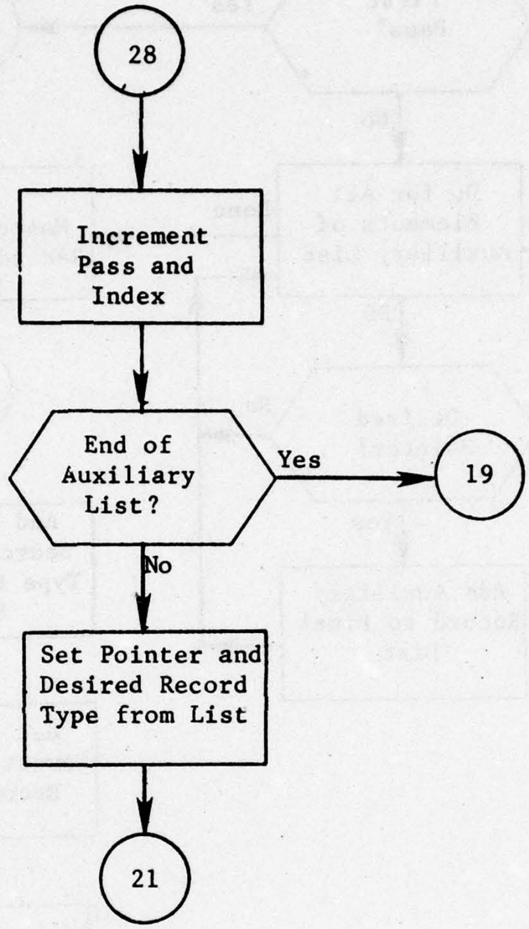


Figure 188. (Part 6 of 8)

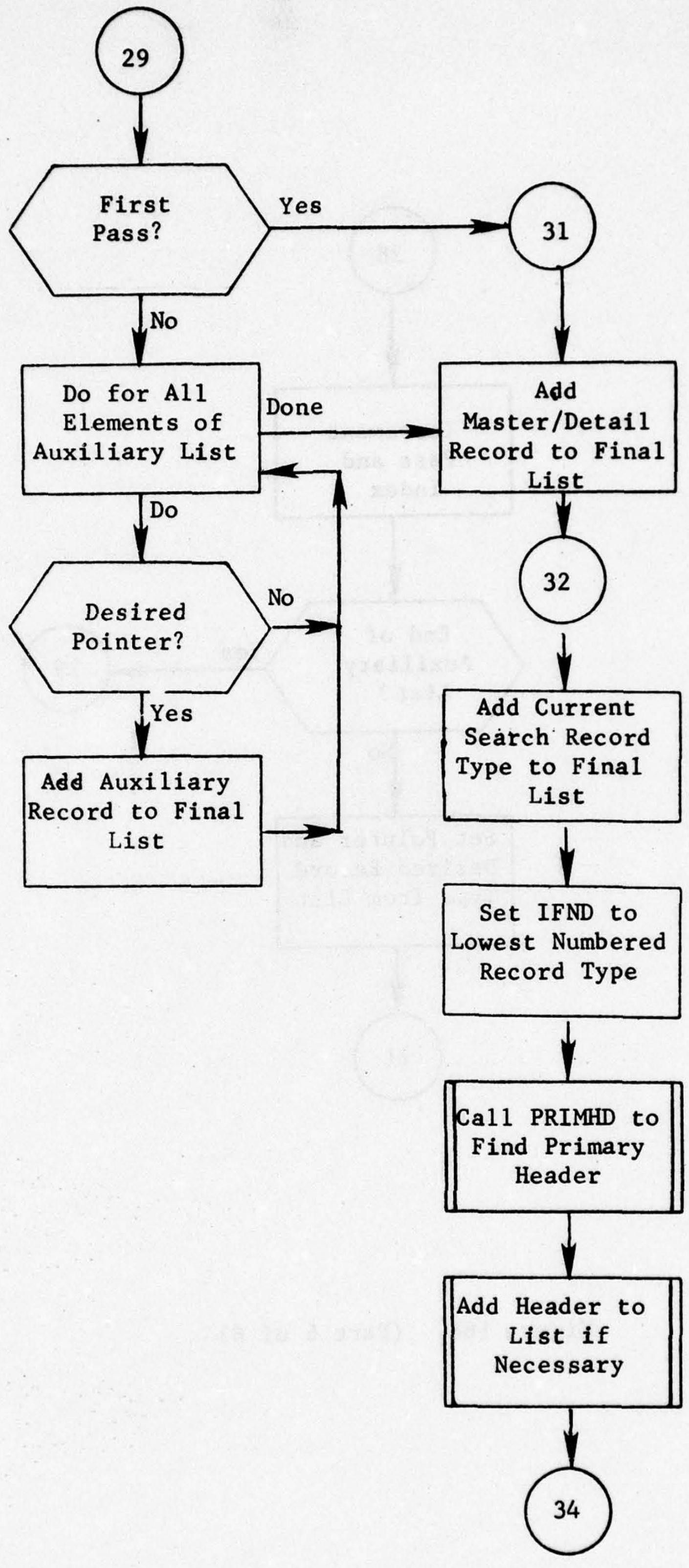


Figure 188. (Part 7 of 8)

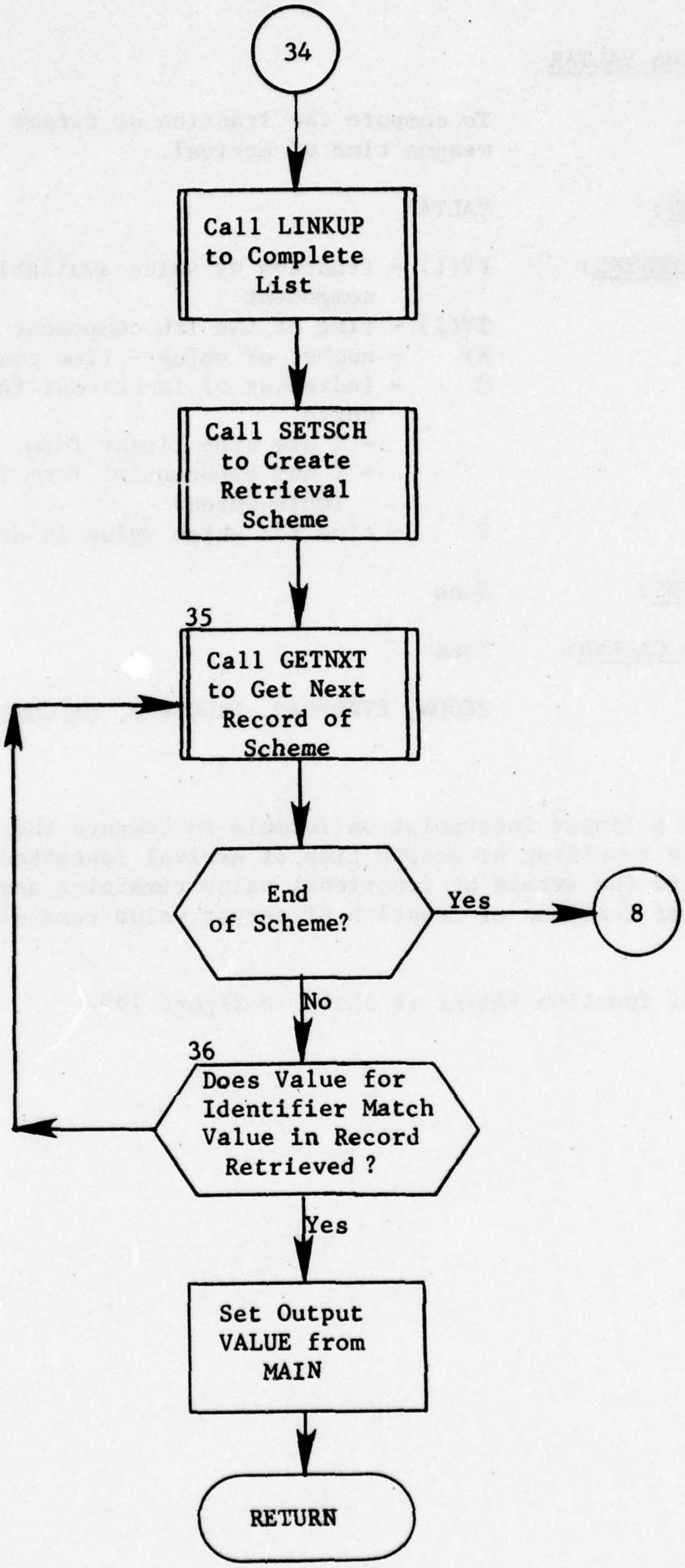


Figure 188. (Part 8 of 8)

9.59 Function VALTAR

PURPOSE: To compute the fraction of target value at weapon time of arrival.

ENTRY POINTS: VALTAR

FORMAL PARAMETERS:

FV(I) - fraction of value available at the Ith component
TV(I) - time of the Ith component
NV - number of value - time components (1-5)
M - indicator of functional form of time curve
= 1 use step-linear form
= 2 use exponential form (not presently implemented)
T - time for which value is desired

COMMON BLOCKS: None

SUBROUTINES CALLED: None

CALLED BY: RECON, EVALPLAN, PROCCOMP, CALCOMP, SALVAL

Method:

VALTAR uses a linear interpolation formula to compute the fraction of target value remaining at weapon time of arrival (parameter T), where FV and TV are the arrays of fractional value remaining and time forming a step-linear function of fraction of target value remaining versus time.

The logic of function VALTAR is shown in figure 189.

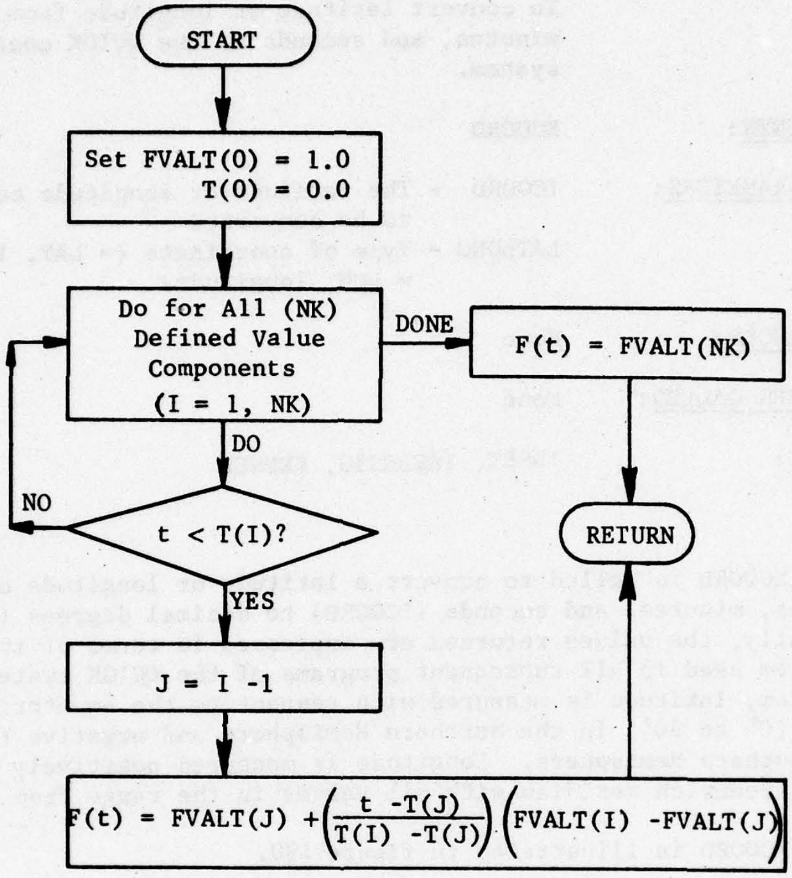


Figure 189. Function VALTAR

9.60 Function XCOORD

PURPOSE: To convert latitude or longitude from degrees, minutes, and seconds to the QUICK coordinate system.

ENTRY POINTS: XCOORD

FORMAL PARAMETERS: ICOORD - The latitude or longitude coordinate to be converted

LATLONG - Type of coordinate (= LAT, latitude; = LON, longitude)

COMMON BLOCKS: None

SUBROUTINES CALLED: None

CALLED BY: INSET, TARDESIG, KRUNCH

Method:

Function XCOORD is called to convert a latitude or longitude coordinate in degrees, minutes, and seconds (ICOORD) to decimal degrees (XCOORD). Additionally, the values returned are expressed in terms of the coordinate system used in all subsequent programs of the QUICK system. In this system, latitude is measured with respect to the equatorial plane, positive (0° to 90°) in the Northern Hemisphere and negative (0° to -90°) in the Southern Hemisphere. Longitude is measured positively westward from the Greenwich meridian with all values in the range from 0° to 360°.

Function XCOORD is illustrated in figure 190.

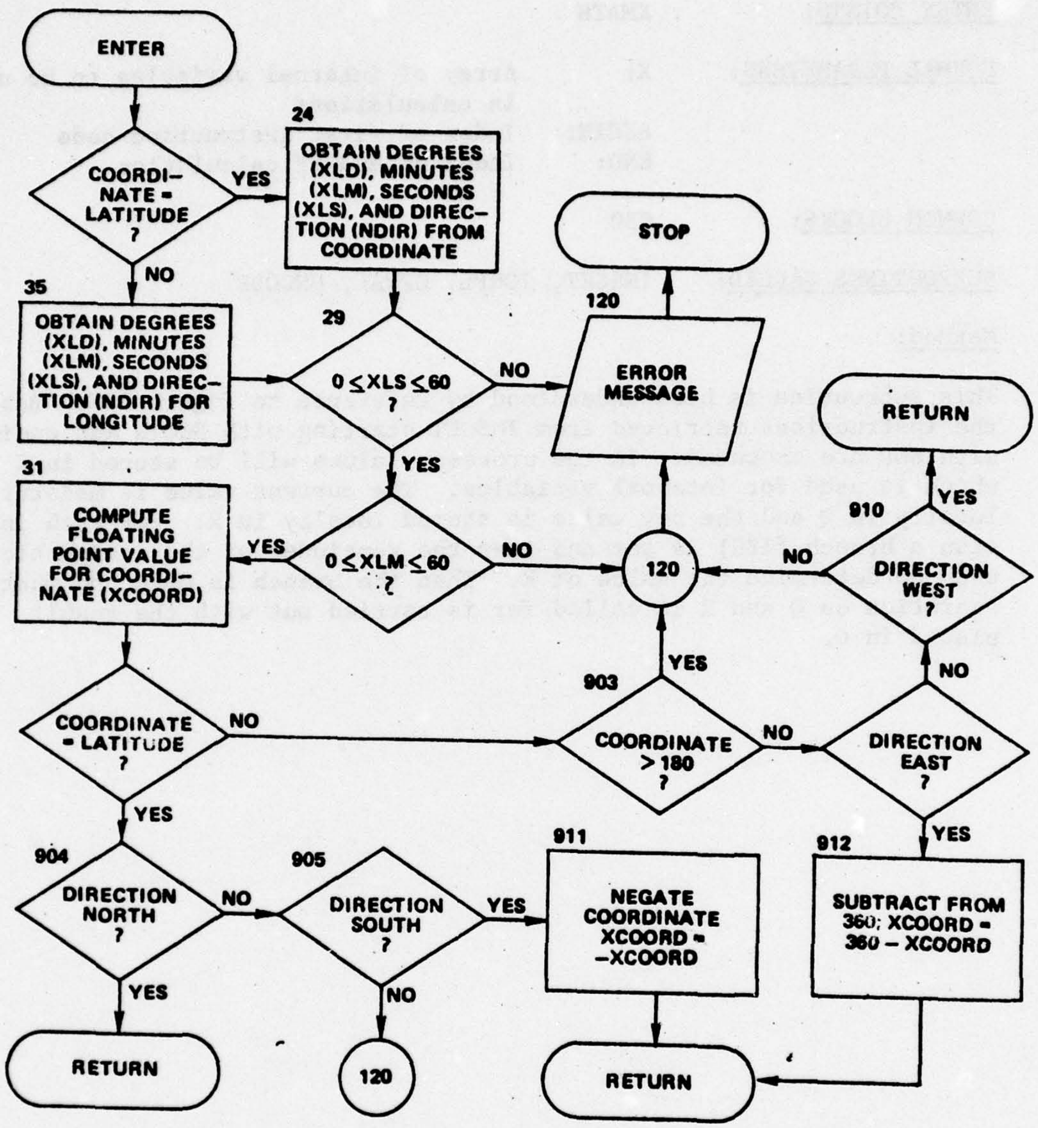


Figure 190. Function XCOORD

9.61 Subroutine XMATH

PURPOSE:	To execute mathematical calculations	
ENTRY POINTS:	XMATH	
FORMAL PARAMETERS:	X:	Array of internal variables to be used in calculations
	BEGIN:	Index of first instruction code
	END:	Index of end of calculation
COMMON BLOCKS:	C30	
SUBROUTINES CALLED:	INSGET, IORFL, OFVAL, UNCODE	

Method:

This subroutine is best understood by reference to figure 191. Basically the instructions retrieved from INSGET starting with BEGIN and ending with END are executed. In the process, values will be stored in X which is used for internal variables. The current value is maintained locally in Q and the new value is stored locally in R. For each instruction a branch (IBR) is set and then the remainder of the instruction is used to determine the value of R. Then the branch is made and whatever operation on Q and R is called for is carried out with the result placed in Q.

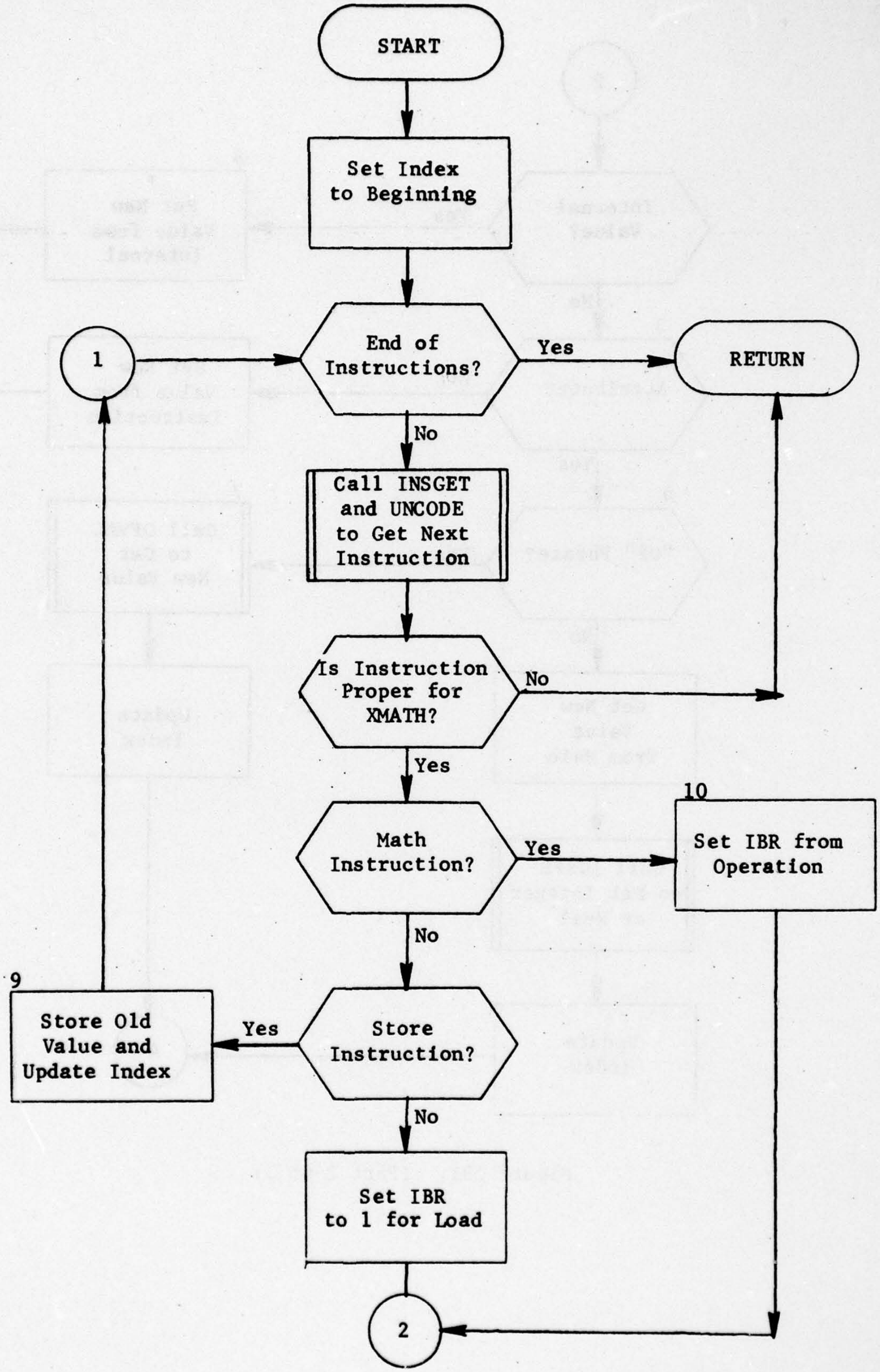


Figure 191. Subroutine XMATH (Part 1 of 3)

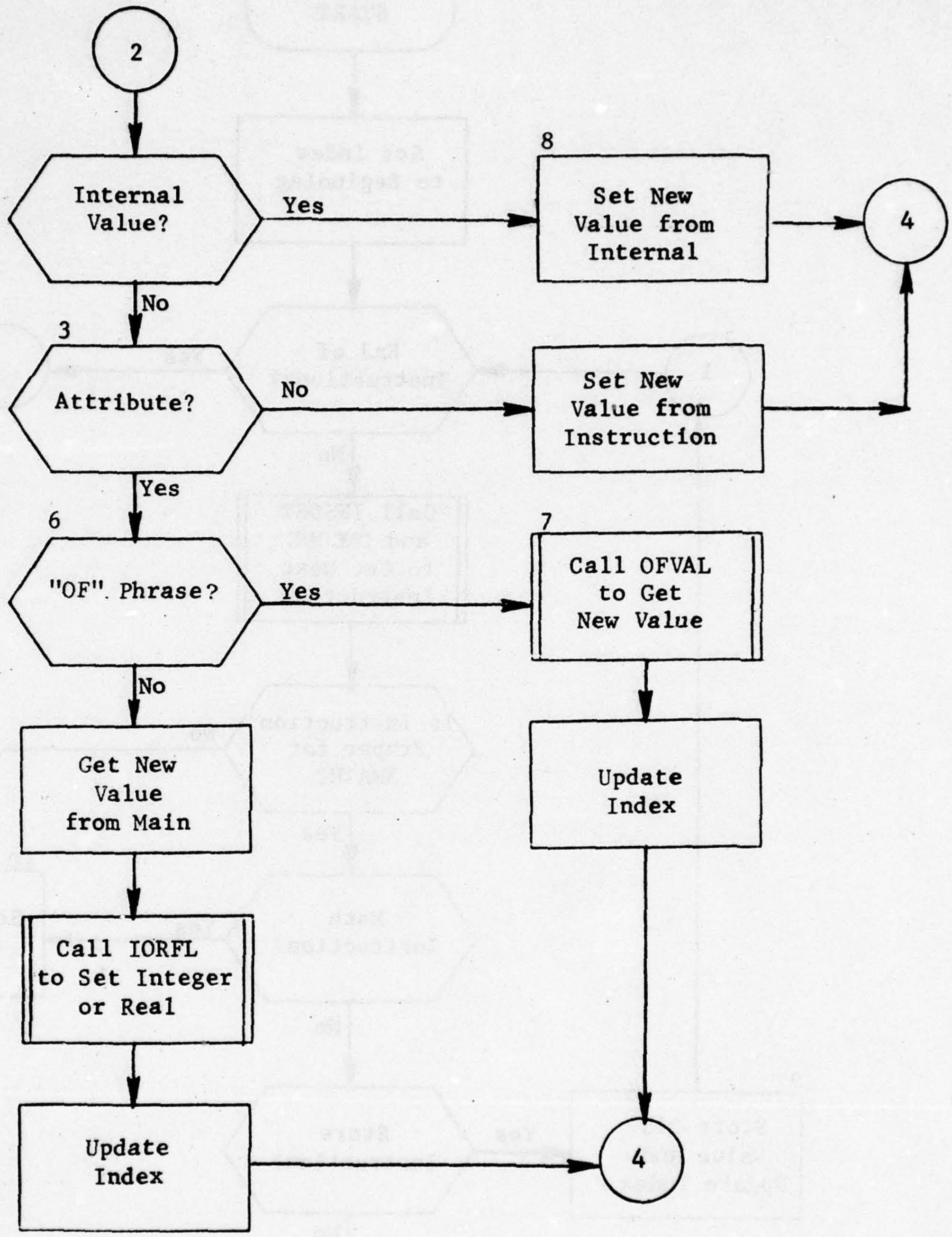


Figure 191. (Part 2 of 3)

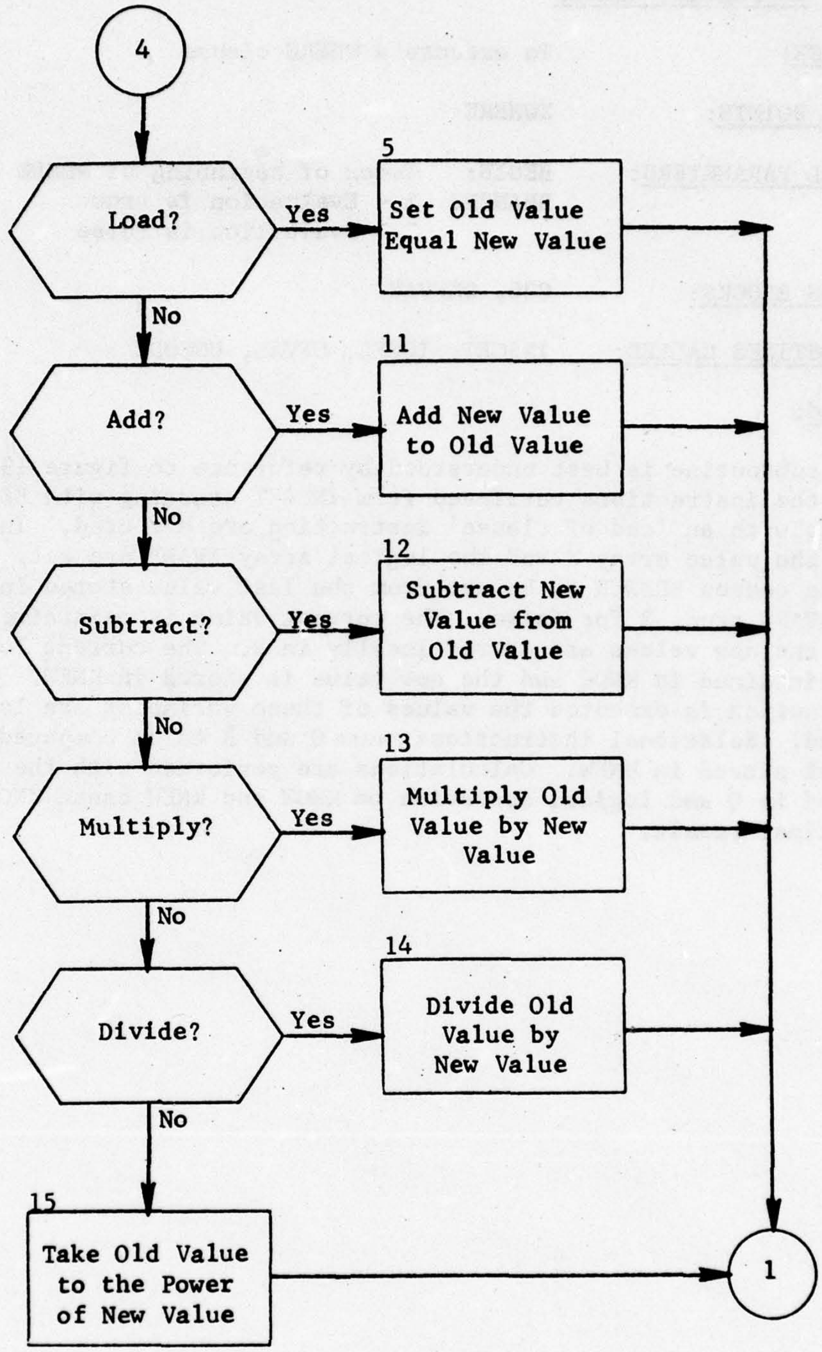


Figure 191. (Part 3 of 3)

9.62 Subroutine XWHERE

PURPOSE:	To execute a WHERE clause
ENTRY POINTS:	XWHERE
FORMAL PARAMETERS:	BEGIN: Index of beginning of WHERE clause BRANCH: 1 - Evaluation is true 2 - Evaluation is false
COMMON BLOCKS:	C30, DEFVAR
SUBROUTINES CALLED:	INSGET, IORFL, OFVAL, UNCODE

Method:

This subroutine is best understood by reference to figure 192. Basically the instructions retrieved from INSGET starting with BEGIN and ending with an 'end of clause' instruction are executed. In the process the value array X and the logical array AVARB are set. The end of clause causes BRANCH to be set from the last value stored in AVARB -- 1 for AVARB true, 2 for false. The current value is maintained locally in Q and new values are stored locally in R. The current logical value is maintained in KNOW and the new value is stored in KNEW. As each instruction is executed the values of these variables are loaded or stored. Relational instructions cause Q and R to be compared with the result placed in KNEW. Calculations are performed with the results placed in Q and logical operation on KNOW and KNEW cause KNOW to have the final result.

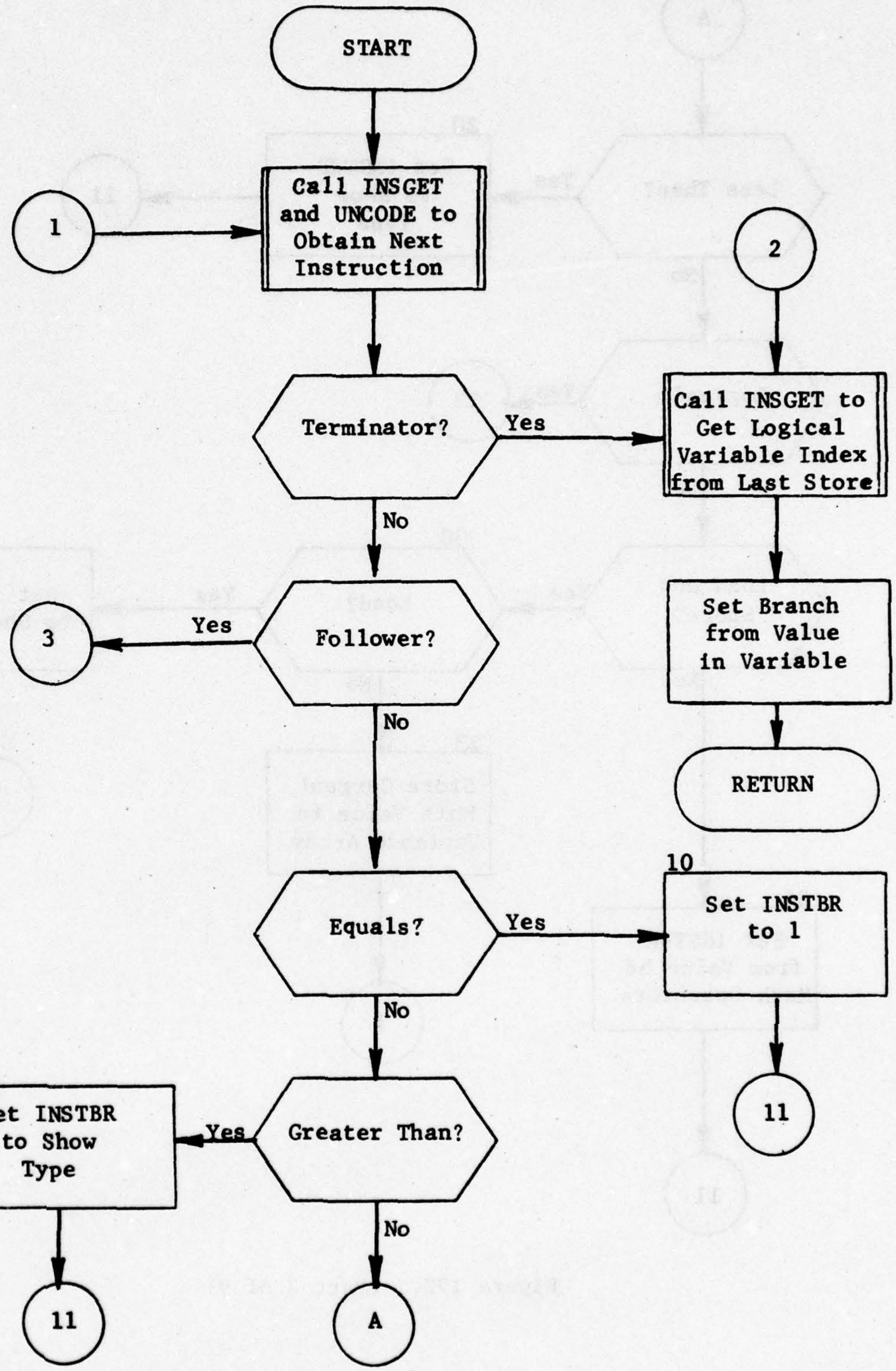


Figure 192. Subroutine XWHERE (Part 1 of 9)

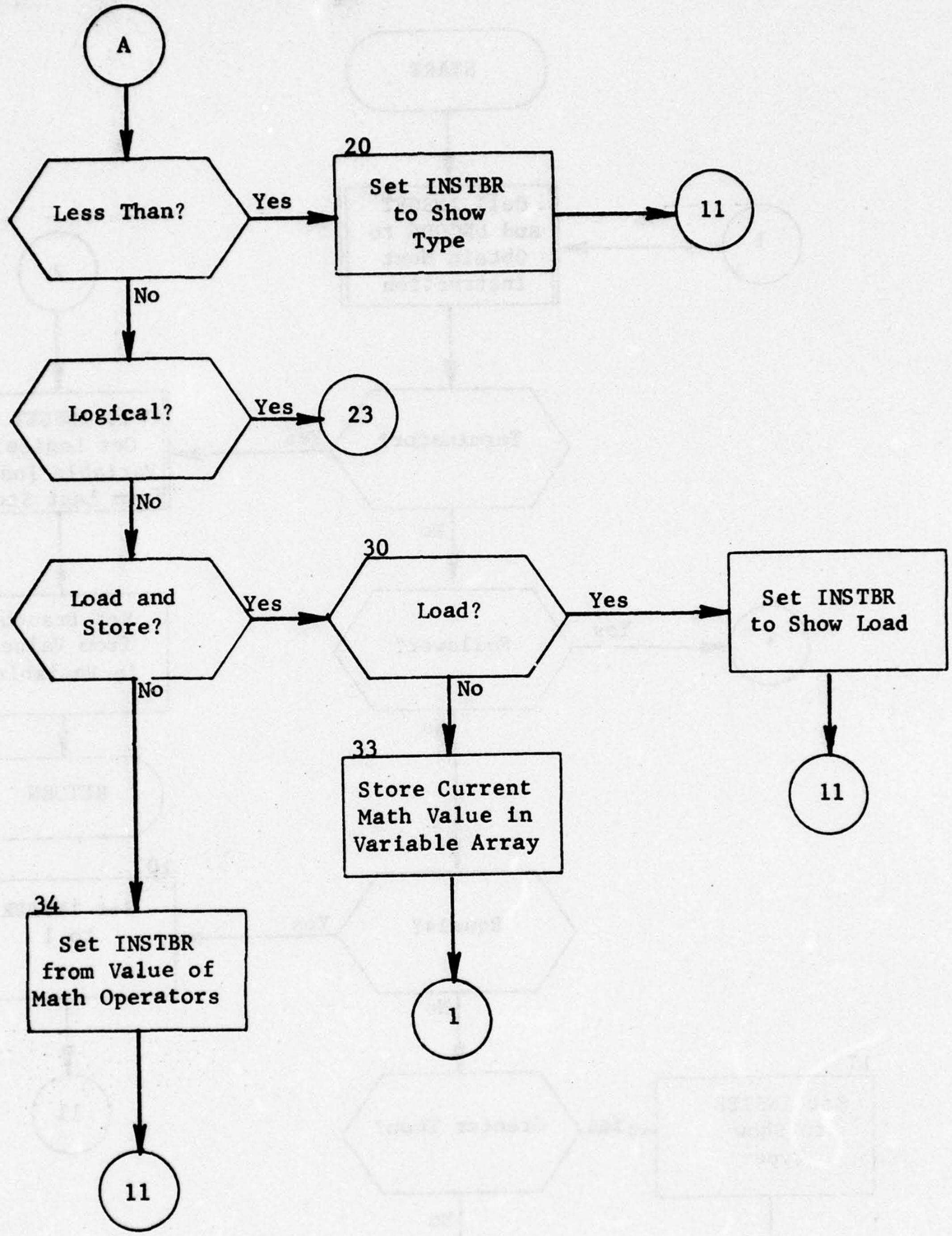


Figure 192. (Part 2 of 9)

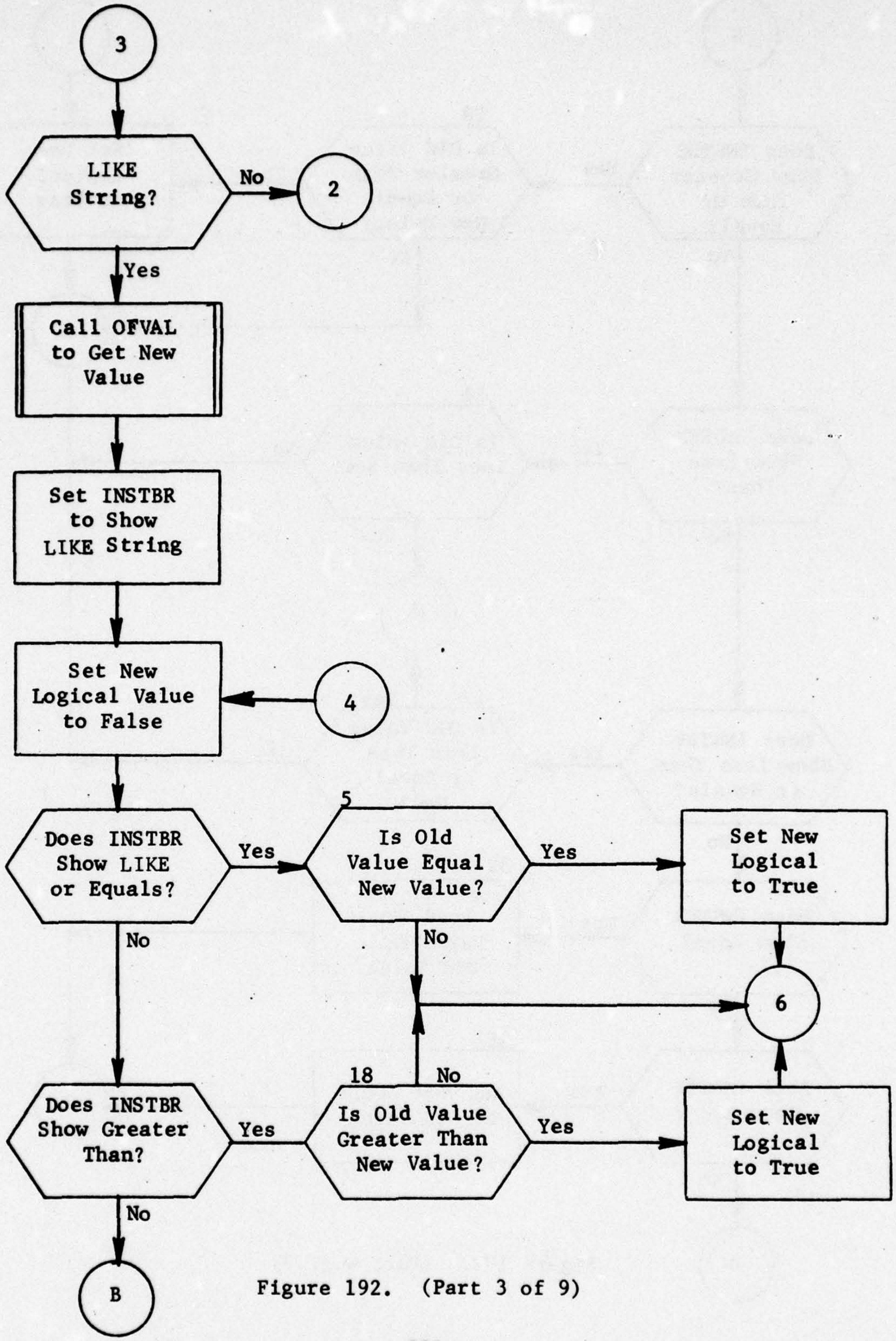


Figure 192. (Part 3 of 9)

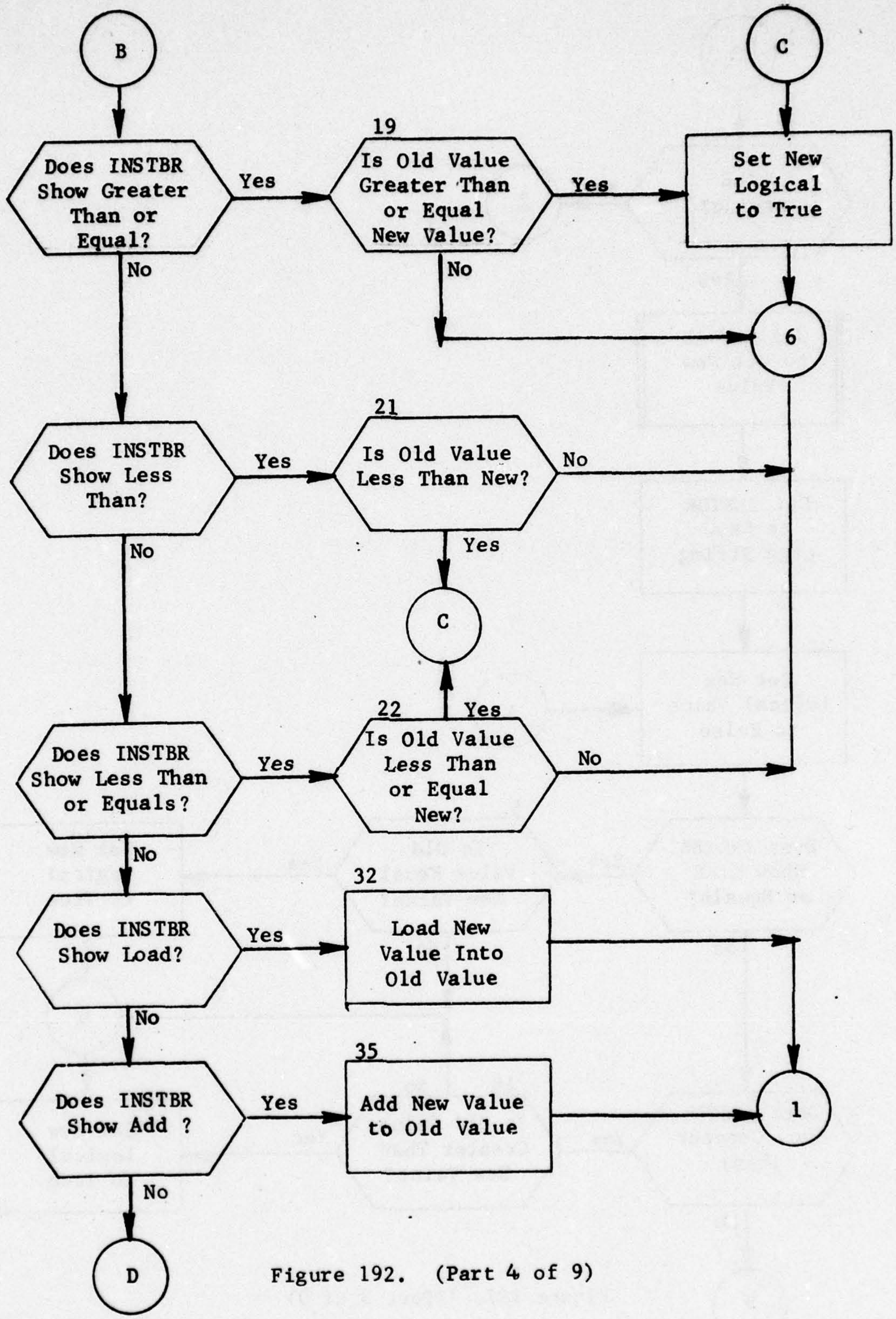


Figure 192. (Part 4 of 9)

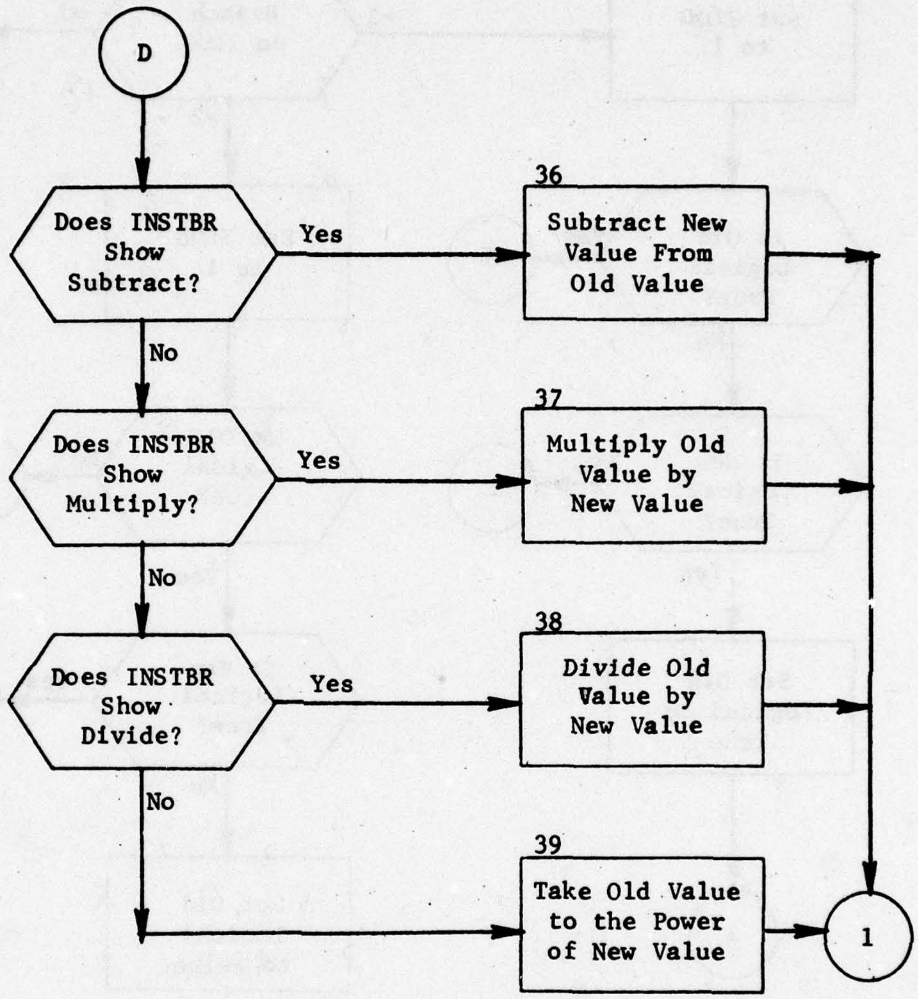


Figure 192. (Part 5 of 9)

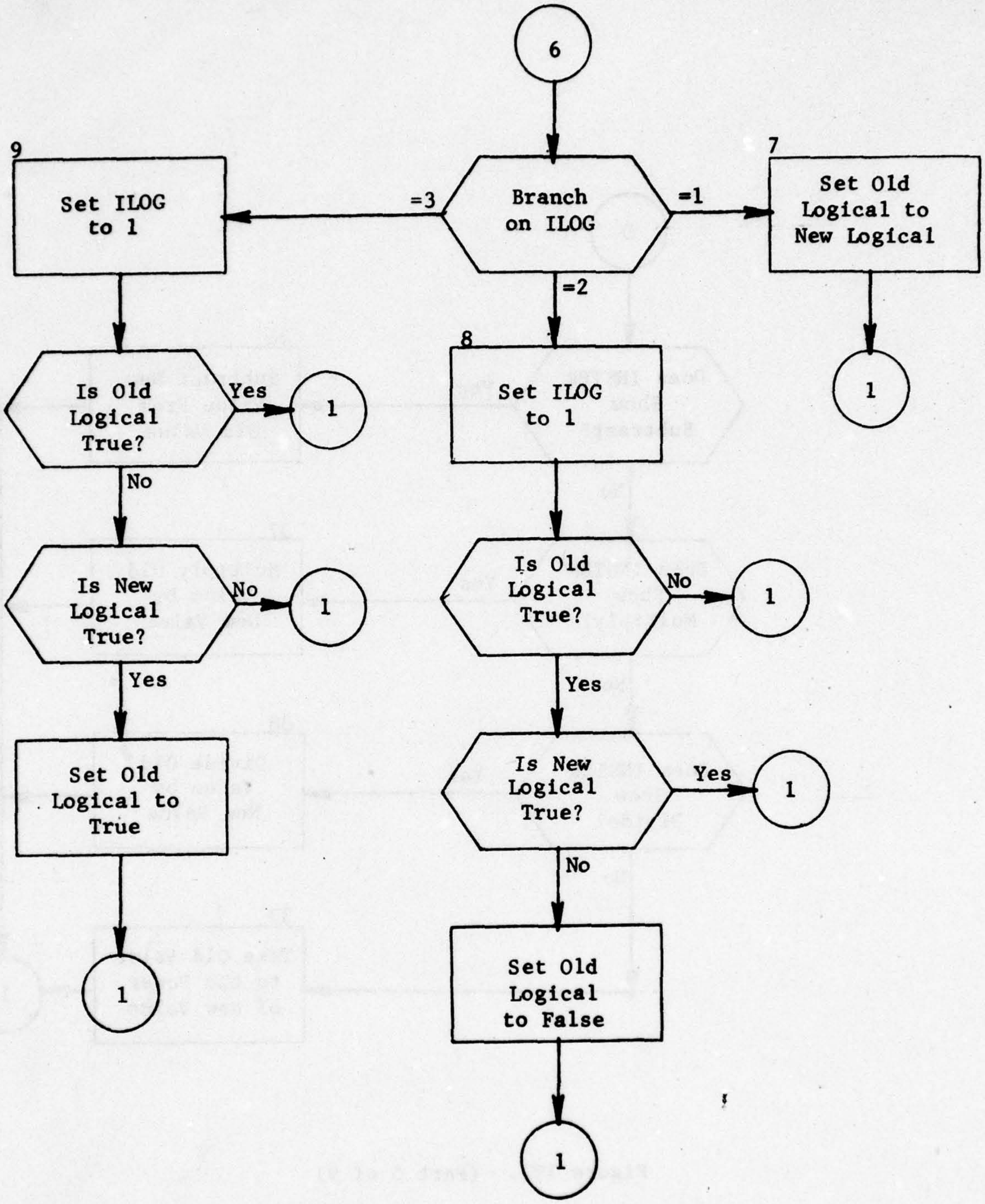


Figure 192. (Part 6 of 9)

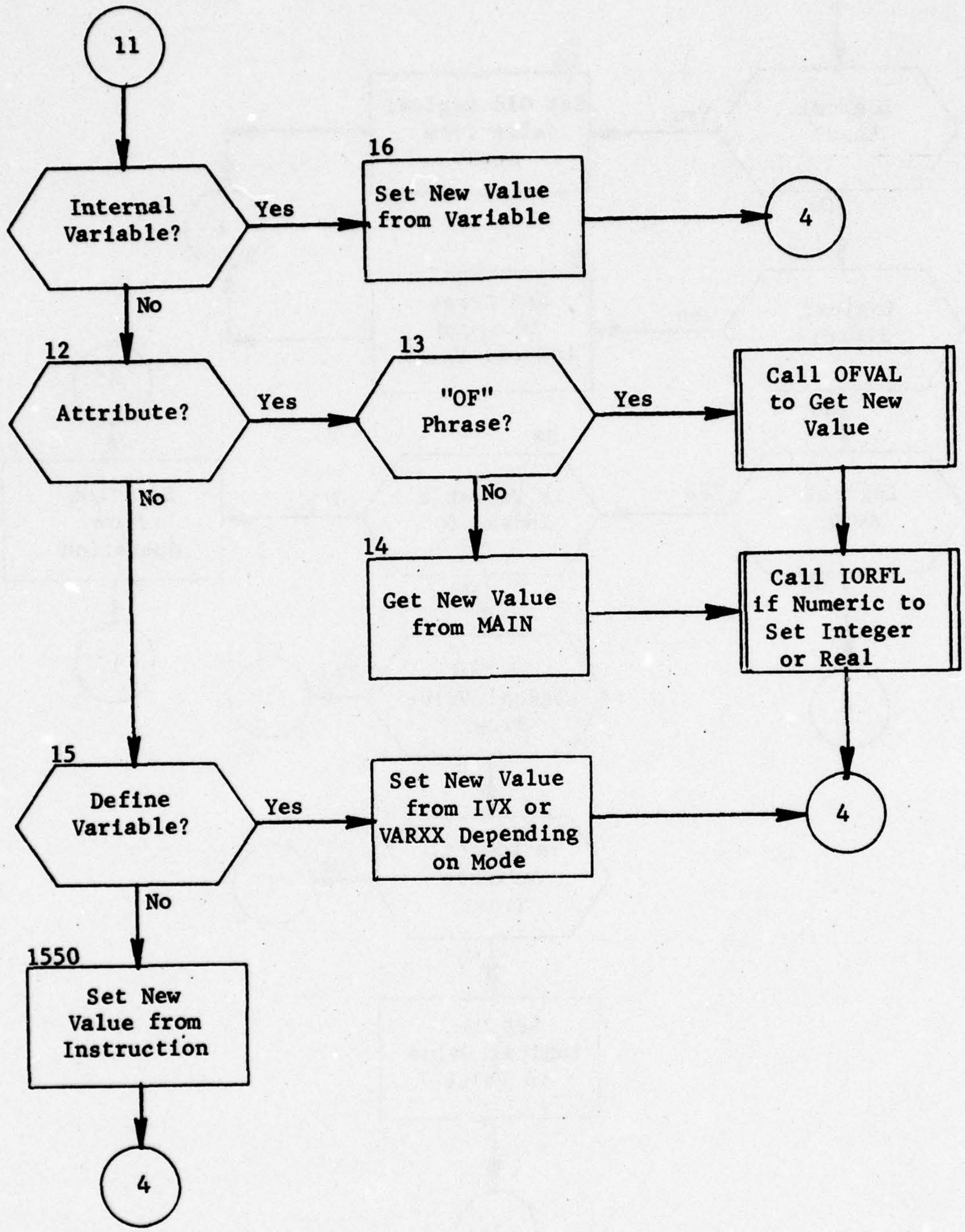


Figure 192. (Part 7 of 9)

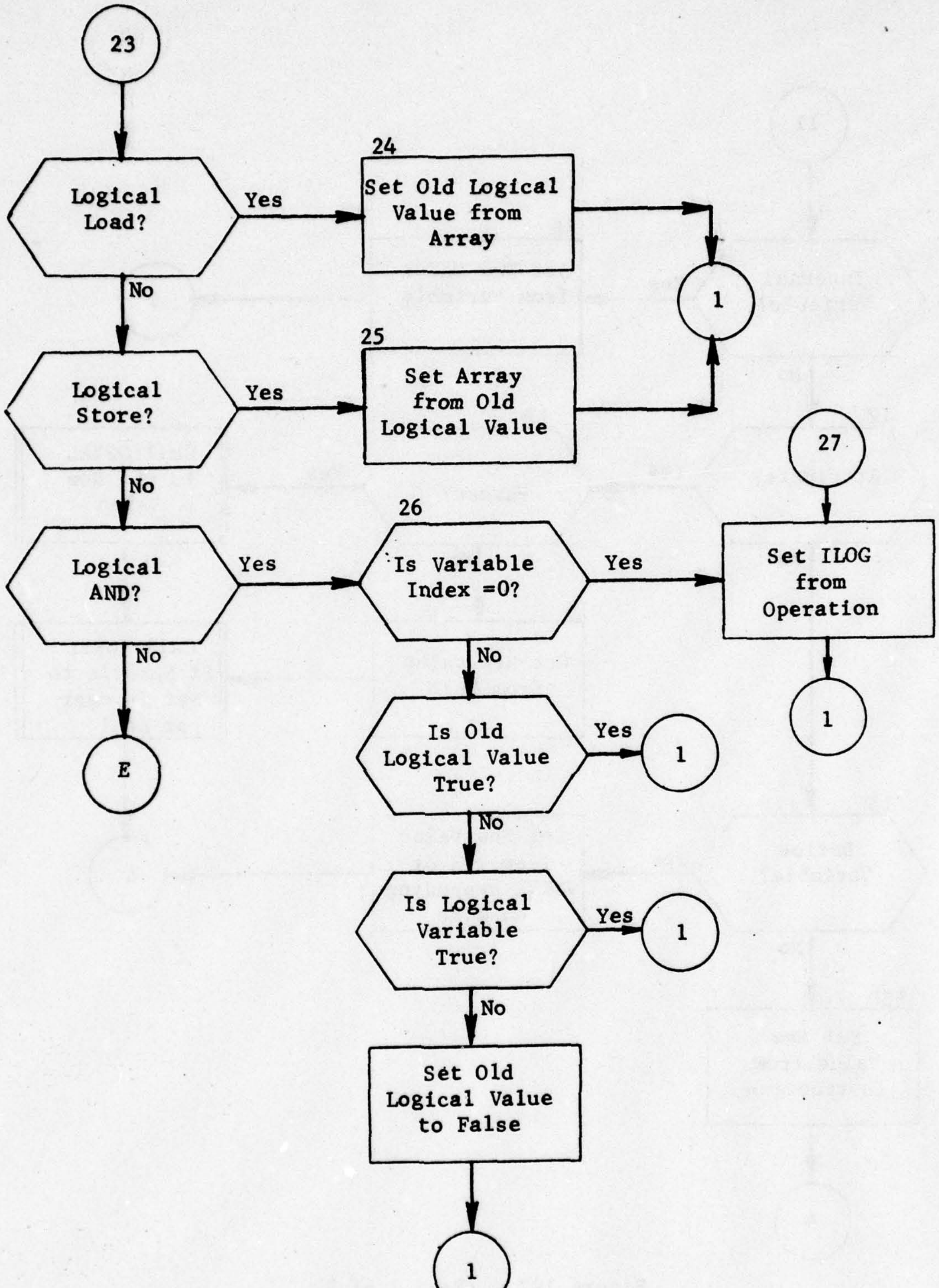


Figure 192. (Part 8 of 9)

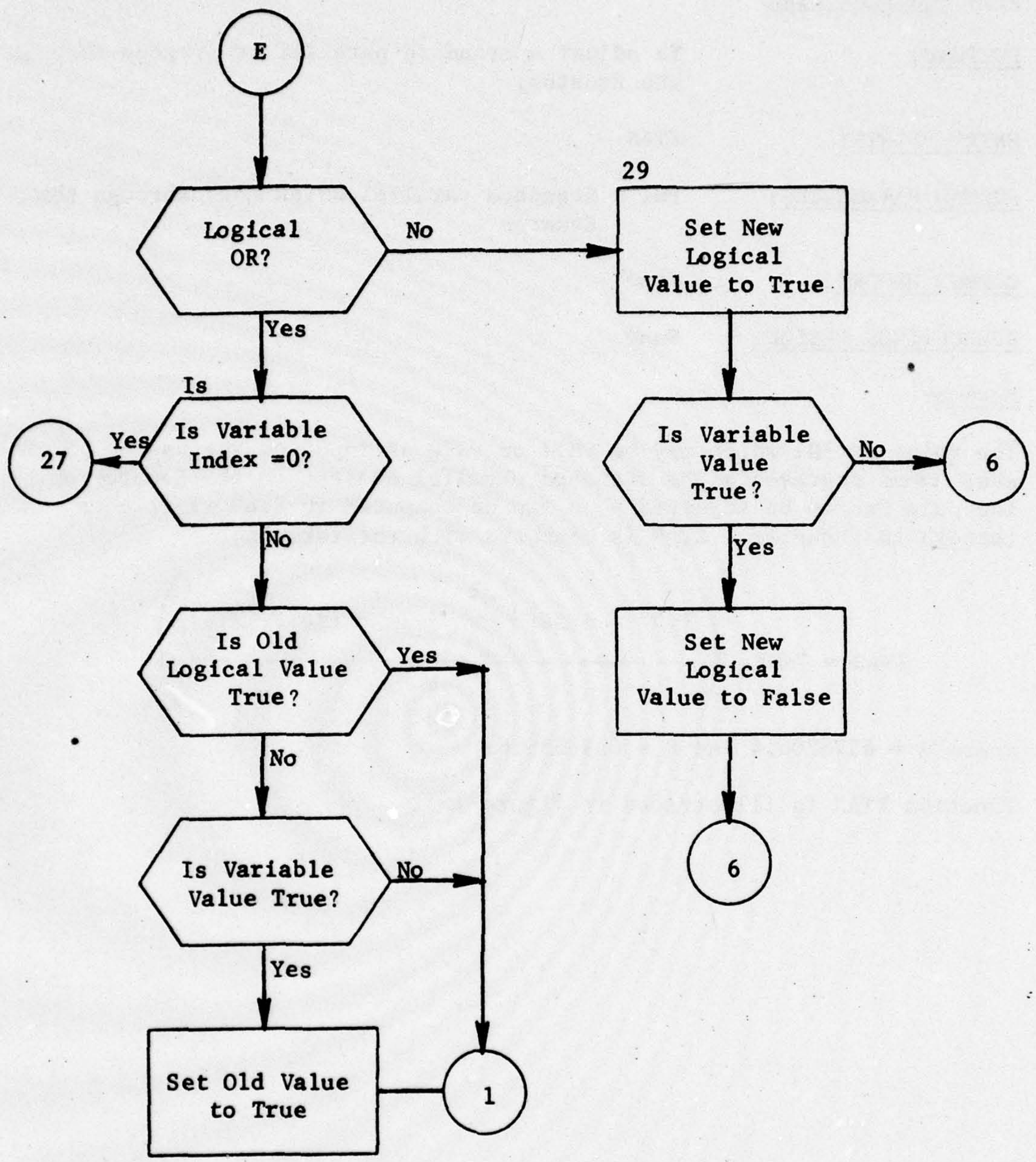


Figure 192. (Part 9 of 9)

9.63 Function ZTAN

PURPOSE: To adjust a standard parallel if it goes through the Equator.

ENTRY POINTS: ZTAN

FORMAL PARAMETERS: PHI - Standard parallel which goes through the Equator

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method:

The value of PHI which may be PHI1 or PHI2 as input on the user's parameter card represents the standard parallel closest to the Equator or the pole has to be adjusted by a factor computed in ZTAN if it goes through the Equator. ZTAN is computed with the formula:

$$\mathrm{ZTAN} = \mathrm{TANF}\left(\frac{1.57 - \mathrm{ATANF}\left(\frac{B^2}{A^2} * \mathrm{TANF}\,(\mathrm{ABSF(PHI)})\right)}{2}\right)$$

where A = 6378206.4 and B = 6356536.8.

Function ZTAN is illustrated in figure 193.

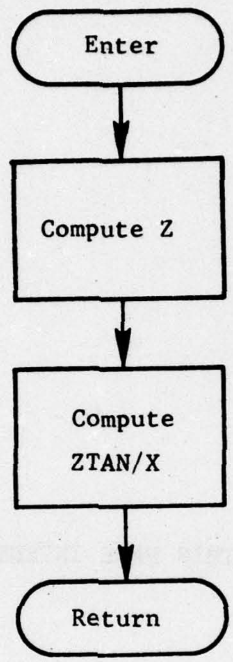


Figure 193. Function ZTAN

Preceding Page BLANK - NOT FILMED

APPENDIX A

COP EXTERNAL COMMON BLOCKS

This appendix contains those common blocks used to communicate between the COP and related modules of the QUICK system. The appendix contains the following common blocks:

o	C10	IDS Communications control block
o	C15	Header reference codes
o	C20	Record type name and number
o	C30	Data base attributes
o	C40	Utility table
o	C50	Display table
o	ERRCOM	IDS error code control block
o	INS	Input instruction code buffer
o	IPGT	Input card image buffer
o	OOPS	Error flag
o	STRING	Interpreted input character string

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
C10	IREFZ	IDS communications control block
	IREFZ	Binary reference code, updated whenever IDS action takes place
	MQ(2)	(Not used)
	IRECTP	Record type number, updated whenever IDS action takes place
	NQ	(Not used)
	ERCODE	IDS error code
	NQ2	(Not used)
C15	HEADRF	Header reference code. BCD representation. Variable is type character *8
C20		Contains values for record type INDRCT
	RNAME	Record type name
	RNUMB	Record type number
C30	MAIN(227)	Contains all data base attributes. For precise attributes definition and their addresses within the array see Users Manual, Volume I
C40		Utility table (TABLEZ)
	TABREF	TABLEZ BCD reference code (type is character*8)
	TABLE(100)	Body of table
C50		Display table (DISPDT)
	DSPTAB(100)	Body of table (see section 6)
ERRCOM	NORMAC	Action to take if error code is not in list CHEKS (ABORT, FLAG, PASS)
	CHCKAC	Action to take if error code is in list CHEKS
	ERUNIT	Unit on which to print error message
	NUMCHK	Number of error codes in CHEKS list
	CHEKS(30)	List of error codes to check

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
INS	INSBUF(100)	Input instruction code buffer
	INSREF(50)	Reference codes for input instruction code tables. (Type is character*8)
	INSTBS	Number of instruction code tables
	INSTCR	Index number of instruction code table currently in buffer
IPQT	POINTER	Points to next character of input card image
	INBUF(80)	Input card image. Each letter is stored in separate word
	SPCIAL	Switch which controls use of '+' and '-' (true if preceding input string was an operator; else false)
	ENDSW	End of input switch
OOPS	ERROR	Error flag, causes COP to check only syntax when on
STRING	TYPE	Current input string's type (not the attribute of the same spelling). If output from subroutine GETSTR results are: =1, operator =2, long string delineates =9, alphabetic =10, numeric
	VALUE	Identifying value, depends on type of string
	NUMBER	Floating point value if string type =10
	ALPHA	Character string being interpreted. Will be blank if string type =1.

APPENDIX B

EXECUTABLE JOB CONTROL LANGUAGE (JCL)
QUICK SYSTEM

The QUICK system executes from an H*(HIS6000 System Loadable file). Figure 194 contains the JCL necessary to build the H* directly from source files. Note that within the FORTY activity for each link an object file (catalog 632IDPOO/QUIK/OBJECT) is also created. Figure 195 contains the JCL to recreate the H* from these object files. In effect, the programmer needs only replace the SELECT of the object deck with the corresponding FORTY activity for each overlay link in which the source has changed to recreate the H*. Finally, figure 196 contains the JCL to create the QUICK utility library from source.

```
$       IDENT    5162,COMPL,314,I M A PROG,631,12
$       USERID   632IDP99$PASSWORD/UZZ
$       LIMITS   30,60K,,50K
$       OPTION   FORTRAN,NOGO
$       LIBRARY  UL,PL
$       LOWLOAD
$       ENTRY    ......
$       FORTY    MAP,XREF,DECK
$       PRMFL    C*,W,S,632IDP00/QUIK/OBJECT/COP
$       SELECT   632IDP00/QUIK/SOURCE/COP/COP
$       SELECT   632IDP00/QUIK/SOURCE/COP/BANNER
$       SELECT   632IDP00/QUIK/SOURCE/COP/ERPROC
$       SELECT   632IDP00/QUIK/SOURCE/COP/HDFND
$       SELECT   632IDP00/QUIK/SOURCE/COP/INICOP
$       SELECT   632IDP00/QUIK/SOURCE/COP/INSPUT
$       SELECT   632IDP00/QUIK/SOURCE/COP/MODGET
$       IDS      DECK
$       LIMITS   30,60K,,50K
$       FILE     *3,X3R,100R
$       PRMFL    C*,W,S,632IDP00/QUIK/OBJECT/QDATA
$       SELECT   632IDP00/QUIK/SOURCE/COP/QDATA
$       SELECT   632IDP00/QUIK/SOURCE/COP/QDATB
$       USE      .QMAX/1/,.QAREA/3126/,.QMIN/1/,.FRRD.
$       LINK     BOOTT
$       FORTY    MAP,XREF,DECK
$       PRMFL    C*,W,S,632IDP00/QUIK/OBJECT/BOOT
$       SELECT   632IDP00/QUIK/SOURCE/BOOT
$       SELECT   632IDP00/QUIK/SOURCE/COP/DCTFND
$       SELECT   632IDP00/QUIK/SOURCE/COP/MNMFND
$       SELECT   632IDP00/QUIK/SOURCE/COP/NUMFND
$       SELECT   632IDP00/QUIK/SOURCE/COP/RNMFND
$       SELECT   632IDP00/QUIK/SOURCE/COP/SEEKER
$       SELECT   632IDP00/QUIK/SOURCE/COP/STRMAK
$       LINK     TABSTR,BOOT
$       USE      TABLZ/568/
$       LINK     ERRF
$       FORTY    MAP,XREF,DECK
$       PRMFL    C*,W,S,632IDP00/QUIK/OBJECT/ERRFND
$       SELECT   632IDP00/QUIK/SOURCE/COP/ERRFND
$       SELECT   632IDP00/QUIK/SOURCE/COP/LNGSTR
$       SELECT   632IDP00/QUIK/SOURCE/COP/SYNTAX
$       SELECT   632IDP00/QUIK/SOURCE/COP/TABINS
$       SELECT   632IDP00/QUIK/SOURCE/COP/WEBSTR
$       LINK     INPT,ERRF
$       FORTY    MAP,XREF,DECK
```

Figure 194. H* Creation From Source (Part 1 of 6)

```
$        PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/INPTRN
$        SELECT  632IDP00/QUIK/SOURCE/COP/INPTRN
$        SELECT  632IDP00/QUIK/SOURCE/COP/DELTAB
$        SELECT  632IDP00/QUIK/SOURCE/COP/INMATH
$        SELECT  632IDP00/QUIK/SOURCE/COP/LINEIO
$        SELECT  632IDP00/QUIK/SOURCE/COP/PARLEV
$        SELECT  632IDP00/QUIK/SOURCE/COP/TABGET
$        LINK    JLM,TABSTR
$        FORTY   MAP,XREF,DECK
$        PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/JLM
$        SELECT  632IDP00/QUIK/SOURCE/JLM/JLM
$        LINK    ASSI
$        FORTY   MAP,XREF,DECK
$        PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/ASSIGN
$        SELECT  632IDP00/QUIK/SOURCE/JLM/ASSIGN
$        SELECT  632IDP00/QUIK/SOURCE/JLM/ALPHAS
$        SELECT  632IDP00/QUIK/SOURCE/JLM/PLAYERS
$        SELECT  632IDP00/QUIK/SOURCE/JLM/TOPRINT
$        LINK    SELE,ASSI
$        FORTY   MAP,XREF,DECK
$        PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/SELECT
$        SELECT  632IDP00/QUIK/SOURCE/JLM/SELECT
$        SELECT  632IDP00/QUIK/SOURCE/JLM/ADTOBASE
$        SELECT  632IDP00/QUIK/SOURCE/JLM/DEFAULT
$        SELECT  632IDP00/QUIK/SOURCE/JLM/KRUNCH
$        SELECT  632IDP00/QUIK/SOURCE/JLM/SAMSET
$        SELECT  632IDP00/QUIK/SOURCE/JLM/SETDEF
$        LINK    ASTE,SELE
$        FORTY   MAP,XREF,DECK
$        PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/ASTERISK
$        SELECT  632IDP00/QUIK/SOURCE/JLM/ASTERISK
$        LINK    DATA,JLM
$        FORTY   MAP,XREF,DECK
$        PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/DATA
$        SELECT  632IDP00/QUIK/SOURCE/DATA/DATA
$        LINK    DATADL
$        FORTY   MAP,XREF,DECK
$        PRMFL   X*,W,S,632IDP00/QUIK/OBJECT/DELETE
$        SELECT  632IDP00/QUIK/SOURCE/DATA/DELETE
$        LINK    DATACH,DATADL
$        FORTY   MAP,XREF,DECK
$        PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/CHANGE
$        SELECT  632IDP00/QUIK/SOURCE/DATA/CHANGE
$        SELECT  632IDP00/QUIK/SOURCE/DATA/DESSCH
$        SELECT  632IDP00/QUIK/SOURCE/DATA/NXTDES
```

Figure 194. (Part 2 of 6)

```
$     SELECT   632IDP00/QUIK/SOURCE/DATA/VALPUT
$     LINK     DATACR,DATACH
$     FORTY    MAP,XREF,DECK
$     PRMFL    C*,W,S,632IDP00/QUIK/OBJECT/CREATE
$     SELECT   632IDP00/QUIK/SOURCE/DATA/CREATE
$     SELECT   632IDP00/QUIK/SOURCE/DATA/VALPUT
$     LINK     DBMOD,DATA
$     FORTY    MAP,XREF,DECK
$     PRMFL    C*,W,S,632IDP00/QUIK/OBJECT/DBMOD
$     SELECT   632IDP00/QUIK/SOURCE/DBMOD/DBMOD
$     SELECT   632IDP00/QUIK/SOURCE/DBMOD/DESTAB
$     LINK     INDXER,DBMOD
$     FORTY    MAP,XREF,DECK
$     PRMFL    C*,W,S,632IDP00/QUIK/OBJECT/INDEXER
$     SELECT   632IDP00/QUIK/SOURCE/INDEXER/INDEXER
$     SELECT   632IDP00/QUIK/SOURCE/INDEXER/COMPLEX
$     SELECT   632IDP00/QUIK/SOURCE/INDEXER/CRTBLE
$     SELECT   632IDP00/QUIK/SOURCE/INDEXER/SETVAL
$     SELECT   632IDP00/QUIK/SOURCE/INDEXER/VLRADI
$     LINK     PLANS,INDEXER
$     FORTY    MAP,XREF,DECK
$     PRMFL    C*,W,S,632IDP00/QUIK/OBJECT/PLANSET
$     SELECT   632IDP00/QUIK/SOURCE/PLANSET/PLANSET
$     SELECT   632IDP00/QUIK/SOURCE/PLANSET/ADJUSTGP
$     SELECT   632IDP00/QUIK/SOURCE/PLANSET/CALCOMP
$     SELECT   632IDP00/QUIK/SOURCE/PLANSET/GRPEM
$     SELECT   632IDP00/QUIK/SOURCE/PLANSET/PRINTGP
$     SELECT   632IDP00/QUIK/SOURCE/PLANSET/SRTTGT
$     SELECT   632IDP00/QUIK/SOURCE/PLANSET/TANKER
$     SELECT   632IDP00/QUIK/SOURCE/PLANSET/VLRADP
$     LINK     PREP,PLANS
$     FORTY    MAP,XREF,DECK
$     PRMFL    C*,W,S,632IDP00/QUIK/OBJECT/PREPALOC
$     SELECT   632IDP00/QUIK/SOURCE/PREPALOC/PREPALOC
$     LINK     A
$     FORTY    MAP,XREF,DECK
$     PRMFL    C*,W,S,632IDP00/QUIK/OBJECT/TARLST
$     SELECT   632IDP00/QUIK/SOURCE/PREPALOC/TARLST
$     LINK     B,A
$     FORTY    MAP,XREF,DECK
$     PRMFL    C*,W,S,632IDP00/QUIK/OBJECT/PARTA
$     SELECT   632IDP00/QUIK/SOURCE/PREPALOC/PARTA
$     SELECT   632IDP00/QUIK/SOURCE/PREPALOC/DEPROUT
$     SELECT   632IDP00/QUIK/SOURCE/PREPALOC/FACTORCG
$     SELECT   632IDP00/QUIK/SOURCE/PREPALOC/FIXWEP
```

Figure 194. (Part 3 of 6)

```
$        SELECT  632IDP00/QUIK/SOURCE/PREPALOC/MAKECHG
$        SELECT  632IDP00/QUIK/SOURCE/PREPALOC/PENROUT
$        LINK    C,B
$        FORTY   MAP,XREF,DECK
$        PRMFL   C*,W,S,632IDP00/QUICK/OBJECT/WEPPREP
$        SELECT  632IDP00/QUIK/SOURCE/PREPALOC/WEPPREP
$        LINK    D,C
$        FORTY   MAP,XREF,DECK
$        FRMFL   C*,W,S,632IDP00/QUIK/OBJECT/TGTPREP
$        SELECT  632IDP00/QUIK/SOURCE/PREPALOC/TGTPREP
$        LINK    E,D
$        FORTY   MAP,XREF,DECK
$        PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/PARTB
$        SELECT  632IDP00/QUIK/SOURCE/PREPALOC/PARTB
$        SELECT  632IDP00/QUIK/SOURCE/PREPALOC/COMPWRIT
$        SELECT  632IDP00/QUIK/SOURCE/PREPALOC/GRPWRIT
$        LINK    EDIT,PREP
$        FORTY   MAP,XREF,DECK
$        PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/EDITDB
$        SELECT  632IDP00/QUIK/SOURCE/EDITDB/EDITDB
$        LINK    ECOUNT
$        FORTY   MAP,XREF,DECK
$        PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/COUNTS
$        SELECT  632IDP00/QUIK/SOURCE/EDITDB/COUNTS
$        LINK    EGENED,ECOUNT
$        FORTY   MAP,XREF,DECK
$        PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/GENEDIT
$        SELECT  632IDP00/QUIK/SOURCE/EDITDB/GENEDIT
$        SELECT  632IDP00/QUIK/SOURCE/EDITDB/BUILDTAB
$        SELECT  632IDP00/QUIK/SOURCE/EDITDB/FORMLOC
$        SELECT  632IDP00/QUIK/SOURCE/EDITDB/SETFLD
$        SELECT  632IDP00/QUIK/SOURCE/EDITDB/SWITH
$        LINK    ENORMA,EGENED
$        FORTY   MAP,XREF,DECK
$        PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/NORMAL
$        SELECT  632IDP00/QUIK/SOURCE/EDITDB/NORMAL
$        LINK    EPROCE,ENORMA
$        FORTY   MAP,XREF,DECK
$        PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/PROCEDIT
$        SELECT  632IDP00/QUIK/SOURCE/EDITDB/PROCEDIT
$        SELECT  632IDP00/QUIK/SOURCE/EDITDB/XWITH
$        LINK    REPORT,EDIT
$        FORTY   MAP,XREF,DECK
$        PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/REPORT
$        SELECT  632IDP00/QUIK/SOURCE/REPORT/REPORT
```

Figure 194. (Part 4 of 6)

```
$     SELECT  632IDP00/QUIK/SOURCE/REPORT/DSPPUT
$     SELECT  632IDP00/QUIK/SOURCE/REPORT/TABMNT
$     LINK    RPTDSN
$     FORTY   MAP,XREF,DECK
$     PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/DESIGN
$     SELECT  632IDP00/QUIK/SOURCE/REPORT/DESIGN
$     LINK    RPTALT,RPTDSN
$     FORTY   MAP,XREF,DECK
$     PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/ALTER
$     SELECT  632IDP00/QUIK/SOURCE/REPORT/ALTER
$     LINK    RPTDMK,RPTALT
$     FORTY   MAP,XREF,DECK
$     PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/DSPMAK
$     SELECT  632IDP00/QUIK/SOURCE/REPORT/DSPMAK
$     LINK    RPTPRN,RPTDMK
$     FORTY   MAP,XREF,DECK
$     PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/PRINT
$     SELECT  632IDP00/QUIK/SOURCE/REPORT/PRINT
$     SELECT  632IDP00/QUIK/SOURCE/REPORT/XDEFN
$     LINK    SRM,REPORT
$     FORTY   MAP,XREF,DECK
$     PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/SRM
$     SELECT  632IDP00/QUIK/SOURCE/SRM/SRM
$     LINK    EIM,SRM
$     FORTY   MAP,XREF,DECK
$     PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/EIM
$     SELECT  632IDP00/QUIK/SOURCE/EIM/EIM
$     SELECT  632IDP00/QUIK/SOURCE/EIM/CONVLL
$     LINK    BSIDAC
$     FORTY   MAP,XREF,DECK
$     PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/SIDAC
$     SELECT  632IDP00/QUIK/SOURCE/EIM/SIDAC
$     LINK    BOTHER,BSIDAC
$     FORTY   MAP,XREF,DECK
$     PRMFL   C*,W,S,632IDP00/QUIK/OBJECT/BLDOTH
$     SELECT  632IDP00/QUIK/SOURCE/EIM/BLDOTH
$     SELECT  632IDP00/QUIK/SOURCE/EIM/XEDEFN
$     LINK    BTABLE,BOTHER
$     FORTY   MAP,XREF,DECK
$     PRMFL   632IDP00/QUIK/OBJECT/TABLE
$     SELECT  632IDP00/QUIK/SOURCE/EIM/TABLE
$     LINK    PLOTTT,BTABLE
$     FORTY   MAP,XREF,DECK
$     DRMFL   632IDP00/QUIK/OBJECT/PLOTDATA
$     SELECT  632IDP00/QUIK/SOURCE/EIM/PLOTDATA
```

Figure 194. (Part 5 of 6)

```
$        SELECT   632IDP00/QUIK/SOURCE/EIM/PICS
$        SELECT   632IDP00/QUIK/SOURCE/EIM/PROJCT
$        EXECUTE  DUMP,DEBUG,NREST,JREST
$        LIMITS   30,60K,-3K,30K
$        FFILE    P*,LGU/(06,42,43,11,12)
$        PRMFL    H*,R/W,R,632IDP00/QUIK/COP/HSTAR
$        PRMFL    QD,R/W,R,632IDP00/QUIK/COP/IDS
$        PRMFL    UL,R/W,R,632IDP00/QUIK/LIBRARY/UTIL
$        PRMFL    PL,R,S,LIBRARY/PLOTLIB
$        FILE     02,X2R,100L
$        FILE     08,X8R,100L
$        FILE     19,X19R,100L
$        TAPE9    20,X20D,,12345,,INPUT-JAD
$        FILE     21,X21R,50L
$        FFILE    21,NBUFFS/2
$        FILE     22,X22R,50L
$        FFILE    22,NBUFFS/2
$        FILE     23,X23R,50L
$        FILE     23,NBUFFS/2
$        FILE     24,X24R,50L
$        FFILE    24,NBUFFS/2
$        FILE     25,X25R,100R
$        FILE     30,X30R,10L
$        TAPE9    31,X31D,,54321,,OUTPUT-SPILLTAPE
$        TAPE9    32,X32D,,54345,,SAVE-RESTORE-TAPE
$        TAPE9    35,X35D,,67890,,OUTPUT-TAPE-1
$        TAPEO    36,X36D,,98765,,OUTPUT-TAPE-2
$        DATA     I*
$        ENDJOB
***EOF
```

Figure 194. (Part 6 of 6)

```
$       IDENT    5162,COMPL,314,I M A PROG,631,12
$       USERID   632IDP99$PASSWORD/UZZ
$       LIMITS   30,60K,,10K
$       OPTION   FORTRAN,NOGO
$       LIBRARY  UL,PL
$       LOWLOAD
$       ENTRY    ......
$       SELECT   632IDP00/QUIK/OBJECT/COP
$       SELECT   632IDP00/QUIK/OBJECT/QDATA
$       USE      .QMAX/1/,.QAREA/3126/,.QMIN/1/,.FRRD.
$       LINK     BOOTT
$       SELECT   632IDP00/QUIK/OBJECT/BOOT
$       LINK     TABSTR,BOOT
$       USE      TABLZ/568/
$       LINK     ERRF
$       SELECT   632IPD00/QUIK/OBJECT/ERRFND
$       LINK     INPT,ERRF
$       SELECT   632IDP00/QUIK/OBJECT/INPTRN
$       LINK     JLM,TABSTR
$       SELECT   632IDP00/QUIK/OBJECT/JLM
$       LINK     ASSI
$       SELECT   632IDP00/QUIK/OBJECT/ASSIGN
$       LINK     SELE,ASSI
$       SELECT   632IDP00/QUIK/OBJECT/SELECT
$       LINK     ASTE,SELE
$       SELECT   632IDP00/QUIK/OBJECT/ASTERISK
$       LINK     DATA,JLM
$       SELECT   632IDP00/QUIK/OBJECT/DATA
$       LINK     DATADL
$       SELECT   632IDP00/QUIK/OBJECT/DELETE
$       LINK     DATACH,DATADL
$       SELECT   632IDP00/QUIK/OBJECT/CHANGE
$       LINK     DATACR,DATACH
$       SELECT   632IDP00/QUIK/OBJECT/CREATE
$       LINK     DBMOD,DATA
$       SELECT   632IDP00/QUIK/OBJECT/DBMOD
$       LINK     INDXER,DBMOD
$       SELECT   632IDP00/QUIK/OBJECT/INDEXER
$       LINK     PLANS,INDXER
$       SELECT   632IDP00/QUIK/OBJECT/PLANSET
$       LINK     PREP,PLANS
$       SELECT   632IDP00/QUIK.OBJECT/PREPALOC
$       LINK     A
$       SELECT   632IDP00/QUIK/OBJECT/TARLST
$       LINK     B,A
```

Figure 195. H* Creation From Object (Part 1 of 3)

```
$        SELECT   632IDP00/QUIK/OBJECT/PARTA
$        LINK     C,B
$        SELECT   632IDP00/QUIK/OBJECT/WEPPREP
$        LINK     D,C
$        SELECT   632IDP00/QUIK/OBJECT/TGTPREP
$        LINK     E,D
$        SELECT   632IDP00/QUIK/OBJECT/PARTB
$        LINK     EDIT,PREP
$        SELECT   632IDP00/QUIK/OBJECT/EDITDB
$        LINK     ECOUNT
$        SELECT   632IDP00/QUIK/OBJECT/COUNTS
$        LINK     EGENED,ECOUNT
$        SELECT   632IDP00/QUIK/OBJECT/GENEDIT
$        LINK     ENORMA,EGENED
$        SELECT   632IDP00/QUIK/OBJECT/NORMAL
$        LINK     EPROCE,ENORMA
$        SELECT   632IDP00/QUIK/OBJECT/PROCEDIT
$        LINK     REPORT,EDIT
$        SELECT   632IDP00/QUIK/OBJECT/REPORT
$        LINK     RPTDSN
$        SELECT   632IDP00/QUIK/OBJECT/DESIGN
$        LINK     RPTALT,RPTDSN
$        SELECT   632IDP00/QUIK/OBJECT/ALTER
$        LINK     RPTDMK,RPTALT
$        SELECT   632IDP00/QUIK/OBJECT/DSPMAK
$        LINK     REPPRN,RPTDMK
$        SELECT   632IDP00/QUIK/OBJECT/PRINT
$        LINK     SRM,REPORT
$        SELECT   632IDP00/QUIK/OBJECT/SRM
$        LINK     EIM,SRM
$        SELECT   632IDP00/QUIK/OBJECT/EIM
$        LINK     BSIDAC
$        SELECT   632IDP00/QUIK/OBJECT/SIDAC
$        LINK     BOTHER,BSIDAC
$        SELECT   632IDP00/QUIK/OBJECT/BLDOTH
$        LINK     BTABLE,BOTHER
$        SELECT   632IDP00/QUIK/OBJECT/TABLE
$        LINK     PLOTTT,BTABLE
$        SELECT   632IDP00/QUIK/OBJECT/PLOTDATA
$        EXECUTE  DUMP,DEBUG,NREST,JREST
$        LIMITS   10,60K,-3K,10K
$        FFILE    PX,LGU/(06,42,43,11,12)
$        PRMFL    H*,R/W,R,632IDP00/QUIK/COP/HSTAR
$        PRMFL    QD,R/W,R,631IDP00/QUIK/COP/IDS
$        PRMFL    UL,R/W,R,631IDP00/QUIK/LIBRARY/UTIL
```

Figure 195. (Part 2 of 3)

```
$        PRMFL    PL,R,S,LIBRARY/PLOTLIB
$        FILE     02,X2R,100L
$        FILE     08,X8R,100L
$        FILE     19,X19R,100L
$        TAPE9    20,X20D,,12345,,INPUT-JAD
$        FILE     21,X21R,50L
$        FFILE    21,NBUFFS/2
$        FILE     22,X22R,50L
$        FFILE    22,NBUFFS/2
$        FILE     23,X23R,50L
$        FFILE    23,NBUFFS/2
$        FILE     24,X24R,50L
$        FFILE    24,NBUFFS/2
$        FILE     25,X25R,100R
$        FILE     30,X30R,10L
$        TAPE9    31,X31D,,54321,,OUTPUT-SPILLTAPE
$        TAPE9    32,X32D,,54345,,SAVE-RESTORE-TAPE
$        TAPE9    35,X35D,,67890,,OUTPUT-TAPE-1
$        TAPE9    36,X36D,,98765,,OUTPUT-TAPE-2
$        DATA     I*
$        ENDJOB
***EOF
```

Figure 195. (Part 3 of 3)

```
$       IDENT     5162,CUTIL,314,I M A PROG,631,12
$       USERID    632IDP99$PASSWORD/UZZ
$       LIMITS    10,32K,,60K
$       FORTY     DECK,MAP,XREF
$       FILE      C*,X15,100L
$       SELECT    632IPD00/QUIK/MSRC/UTIL/ABORT
$       SELECT    632IDP00/QUIK/MSRC/UTIL/ACOS
$       SELECT    632IDP00/QUIK/MSRC/UTIL/ASIN
$       SELECT    632IDP00/QUIK/SOURCE/UTIL/ATFNDR
$       SELECT    632IDP00/QUIK/MSRC/UTIL/ATN2PI
$       SELECT    632IDP00/QUIK/MSRC/UTIL/BUFIN
$       SELECT    632IDP00/QUIK/SOURCE/UTIL/CINSGET
$       SELECT    632IDP00/QUIK/MSRC/UTIL/CLOSPIL
$       SELECT    632IDP00/QUIK/MSRC/UTIL/CLRMON
$       SELECT    632IDP00/QUIK/MSRC/UTIL/COT
$       SELECT    632IDP00/QUIK/MSRC/UTIL/DELLONG
$       SELECT    632IDP00/QUIK/MSRC/UTIL/DIFFLONG
$       SELECT    632IDP00/QUIK/MSRC/UTIL/DISTF
$       SELECT    632IDP00/QUIK/MSRC/UTIL/DOTLINE
$       SELECT    632IDP00/QUIK/MSRC/UTIL/ENDTAPE
$       SELECT    632IDP00/QUIK/MSRC/UTIL/ERAZE
$       SELECT    632IDP00/QUIK/MSRC/UTIL/FILEHNR
$       SELECT    632IDP00/QUIK/SOURCE/FINDCLAS
$       SELECT    632IDP00/QUIK/SOURCE/UTIL/FINDSIDE
$       SELECT    632IDP00/QUIK/SOURCE/UTIL/FORMAK
$       GMAP      DECK
$       FILE      C*,X1S
$       SELECT    632IDP00/QUIK/MSRC/UTIL/FYLCHK
$       GMAP      DECK
$       FILE      C*,X1S
$       SELECT    632IDP00/QUIK/MSRC/UTIL/GETCLOCK
$       FORTY     DECK,MAP,XREF
$       FILE      C*,X1S
$       SELECT    632IDP00/QUIK/MSRC/UTIL/GETDATE
$       SELECT    632IDP00/QUIK/MSRC/UTIL/GETDF
$       SELECT    632IDP00/QUIK/MSRC/UTIL/GETLOC
$       SELECT    632IDP00/QUIK/SOURCE/UTIL/GETNXT
$       SELECT    632IDP00/QUIK/SOURCE/UTIL/GETSTR
$       SELECT    632IDP00,QUIK/SOURCE/UTIL/GETTAR
$       SELECT    632IDP00/QUIK/MSRC/UTIL/GETVALU
$       SELECT    632IDP00/QUIK/MSRC/UTIL/GLOG
$       SELECT    632IDP00/QUIK/MSRC/UTIL/HOUSKEEP
$       SELECT    632IDP00/QUIK/MSRC/UTIL/IFUNIT
$       SELECT    632IDP00/QUIK/MSRC/UTIL/IGET
$       SELECT    632IDP00/QUIK/MSRC/UTIL/IGETHOB
```

Figure 196. Utility Library Creation (Part 1 of 4)

AD-A054 310 COMMAND AND CONTROL TECHNICAL CENTER WASHINGTON D C F/G 15/7
THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK) PROGR--ETC(U)
JUN 77 D J SANDERS, P F MAYKRANTZ, J M HERRIN
UNCLASSIFIED CCTC-CSM-MM-9-77-V1-PT-2 SBIE-AD-E100 052 NL

6 OF 6
AD
A054310

END
DATE
FILMED
6-78
DDC

```
$          SELECT   632IDP00/QUIK/MSRC/UTIL/IHASH
$          SELECT   632IDP00/QUIK/MSRC/UTIL/INBUFDK
$          SELECT   632IDP00/QUIK/MSRC/UTIL/INBUFTP
$          SELECT   632IDP00/QUIK/MSRC/UTIL/INERRDK
$          SELECT   632IDP00/QUIK/MSRC/UTIL/INERRTP
$          SELECT   632IDP00/QUIK/MSRC/UTIL/INLABEL
$          SELECT   632IDP00/QUIK/MSRC/UTIL/INTERP
$          SELECT   632IDP00/QUIK/MSRC/UTIL/INTERPGC
$          SELECT   632IDP00/QUIK/MSRC/UTIL/INTPIECE
$          SELECT   632IDP00/QUIK/SOURCE/UTIL/IORFL
$          SELECT   632IDP00/QUIK/MSRC/UTIL/IPUT
$          SELECT   632IDP00/QUIK/MSRC/UTIL/ISOFF
$          SELECT   632IDP00/QUIK/MSRC/UTIL/ITLE
$          SELECT   632IDP00/QUIK/MSRC/UTIL/IWANT
$          SELECT   632IDP00/QUIK/MSRC/UTIL/KEYMAKE
$          SELECT   632IDP00/QUIK/MSRC/UTIL/LATBT
$          SELECT   632IDP00/QUIK/SOURCE/UTIL/LINKUP
$          GMAP     DECK
$          FILE     C*,X1S
$          SELECT   631IDP00/QUIK/MSRC/UTIL/LOCF
$          FORTY    DECK,MAP,XREF
$          FILE     C*,X1S
$          SELECT   632IDP00/QUIK/MSRC/UTIL/LOCREAD
$          SELECT   632IDP00/QUIK/MSRC/UTIL/LREORDER
$          SELECT   632IDP00/QUIK/MSRC/UTIL/MAPEDGE
$          SELECT   632IDP00/QUIK/MSRC/UTIL/NEXTAPE
$          SELECT   632IDP00/QUIK/MSRC/UTIL/MODIRC
$          SELECT   632IDP00/QUIK/MSRC/UTIL/NUMDEL
$          SELECT   632IDP00/QUIK/MSRC/UTIL/NUMGET
$          SELECT   632IDP00/QUIK/SOURCE/UTIL/OFVAL
$          SELECT   632IDP00/QUIK/MSRC/UTIL/OPENSPL
$          SELECT   632IDP00/QUIK/MSRC/UTIL/ORDER
$          SELECT   632IDP00/QUIK/MSRC/UTIL/OUTBFDK
$          SELECT   632IDP00/QUIK/MSRC/UTIL/OUTBFTP
$          SELECT   632IDP00/QUIK/MSRC/UTIL/OUTDF
$          SELECT   632IDP00/QUIK/MSRC/UTIL/OUTERDK
$          SELECT   632IDP00/QUIK/MSRC/UTIL/OUTERTP
$          SELECT   632IDP00/QUIK/MSRC/UTIL/PAGESKP
$          SELECT   632IDP00/QUIK/MSRC/UTIL/PIECEIT
$          SELECT   632IDP00/QUIK/MSRC/UTIL/PIECENUM
$          SELECT   632IDP00/QUIK/MSRC/UTIL/PLTTIC
$          SELECT   632IDP00/QUIK/SOURCE/UTIL/PRIMHD
$          SELECT   632IDP00/QUIK/SOURCE/UTIL/PSREC
$          SELECT   632IDP00/QUIK/MSRC/UTIL/RANF
$          SELECT   632IDP00/QUIK/MSRC/UTIL/RANFSET
```

Figure 196. (Part 2 of 4)

```
$      SELECT  632IDP00/QUIK/MSRC/UTIL/RDARRAY
$      SELECT  632IDP00/QUIK/MSRC/UTIL/REORDER
$      SELECT  632IDP00/QUIK/MSRC/UTIL/RLBRT
$      SELECT  632IDP00/QUIK/MSRC/UTIL/SETHEAD
$      GMAP    DECK
$      FILE    C*,X1S
$      SELECT  632IDP00/QUIK/MSRC/UTIL/SETORD
$      FORTY   DECK,AMP,XREF
$      FILE    C*,X1S
$      SELECT  632IDP00/QUIK/MSRC/UTIL/SETREAD
$      SELECT  632IDP00/QUIK/SOURCE/UTIL/SETSCH
$      SELECT  632IDP00/QUIK/MSRC/UTIL/SKIP
$      SELECT  632IDP00/QUIK/MSRC/UTIL/SLOG
$      SELECT  532IDP00/QUIK/SOURCE/UTIL/SORTIT
$      SELECT  632IDP00/QUIK/MSRC/UTIL/SSKPC
$      SELECT  632IDP00/QUIK/MSRC/UTIL/STD
$      GMAP    DECK
$      FILE    C*,X1S
$      SELECT  632IDP00/QUIK/SOURCE/UTIL/SVTP
$      FORTY   DECK,MAP,XREF
$      FILE    C*,X1S
$      SELECT  632IDP00/QUIK/MSRC/UTIL/TAN
$      SELECT  632IDP00/QUIK/MSRC/UTIL/TERMTAP
$      SELECT  632IDP00/QUIK/MSRC/UTIL/TICMAKE
$      SELECT  632IDP00/QUIK/MSRC/UTIL/TIMEDAY
$      SELECT  632IDP00/QUIK/MSRC/UTIL/TIMEME
$      SELECT  632IDP00/QUIK/MSRC/UTIL/TOFM
$      SELECT  632IDP00/QUIK/SOURCE/UTIL/UNCODE
$      SELECT  632IDP00/QUIK/SOURCE/UTIL/VALFND
$      SELECT  632IDP00/QUIK/MSRC/UTIL/VALTAR
$      SELECT  632IDP00/QUIK/MSRC/UTIL/XCOORD
$      SELECT  632IDP00/QUIK/SOURCE/UTIL/XMATH
$      SELECT  632IDP00/QUIK/SOURCE/UTIL/XWHERE
$      SELECT  632IDP00/QUIK/MSRC/UTIL/ZTAN
$      UTILITY
$      LIMITS  10,32K,,60K
$      FUTIL   BB,CC,RWD/BB/,COPY/1R/
$      FUTIL   AA,CC,RWD/AA/,COPY/1F/
$      FUTIL   BB,CC,RWD/BB/,SKIP/1R/,COPY/1R/,RWD/CC/
$      FILE    AA,X1S,100L
$      FILE    CC,X4S,100L
$      DATA    BB,,COPY
$      INCLUDE
$      ENDEDIT
$      ENDCOPY
```

Figure 196. (Part 3 of 4)

```
$       FILEDIT NOSOURCE,OBJECT,INITIALIZE
$       FILE    R*,X2S,100L
$       FILE    *C,X4R,100L
$       PROGRAM RANLIB
$       FILE    R*,X2R,100L
$       PRMFL   A4,R/W,R,632IDP00/QUIK/LIBRARY/UTIL
$       ENDJOB
```

Figure 196. (Part 4 of 4)

APPENDIX C

PERFORM PROGRAM

This appendix contains maintenance information for the PERFORM program. PERFORM is an on-line program designed to generate remote job entry jobs for the QUICK system.

C.1 Purpose

PERFORM is an on-line interactive program which creates a file of Job Control Language (JCL) according to user directions. This file of JCL may be set up to perform any combination of the following functions:

- o Run QUICK
- o Recompile a module of QUICK and recreate the QUICK H* file
- o Recompile and recreate the QUICK uttility subroutine library

C.2 Input

PERFORM is an interactive system and, therefore, obtains part of its input from user responses from the terminal. PERFORM also has three files (or sets of files) which it uses to build the job stream JCL.

- o A set of files each of which contains the JCL required to define a module and its linkage from object files. These files are all under the catalogue 631IDPOO/TEST/COP/CANOF

- o A set of files each of which contains the JCL required to define a module and its linkage from source files. These files are all under this catalogue 631IDPOO/TEST/COP/NEWCANOF

- o A file (631IDPOO/PERFORM/VRBLIM) which details the various required limits for the system and which contains one record for each legal verb:

  - Word 1: Verb Name
  - Word 2: Maximum CPU Time
  - Word 3: Maximum Core Requirements
  - Word 4: Maximum Lines of Output

C.3 Output

As output, PERFORM creates a file of JCL in ASCI format. This file (631IDPOO/THEJOB) is left in the Active File Table (AFT) so that the user may immediately submit the job using the CARDIN subsystem.

C.4 Concept of Operation

PERFORM follows a series of steps at user direction. For details of the question and answer sequences see *Users Manual, Volume I*. Based on the selection of 'MODE,' PERFORM generates either the compile or the run JCL on file THE JOB or both. For the compile JCL, PERFORM asks for a list of the modules changed. Then PERFORM goes through the list of all modules. For each module in the list of those changed, PERFORM adds a file from the NEWCANOF catalogue. For each module not in the list of changed modules, PERFORM adds a file from the CANOF catalogue. If 'UTILITY' is selected as a module change, the file CANOF/UTIL is added also. After all compile JCL has been setup, the user is asked if a run is also desired.

For the run mode, the file CANOF/RUN is added. Then the user supplies a list of the verbs being used. From this list, PERFORM uses the information stored on file VRBLIM to compute the LIMITS card parameters. The user may alter these. Then the user is requested to add data files and/or lines of input. Finally, the LIMITS card and other final cards are added and the user is instructed as to how to submit the job.

C.5 Major Subroutine

PERFORM has only one subroutine READIN. This subroutine scans the user input and converts any lower case ASCI letters to upper case ASCI letters.

C.6 Common Blocks

PERFORM has only one common block LINE. This block contains the array LINE(80) which is used to communicate with subroutine READIN. Each word of LINE contains one character of input.

C.7 Program PERFORM

PURPOSE: To build JCL through on-line interaction

ENTRY POINTS: ......

FORMAL PARAMETERS: None

COMMON BLOCKS: LINE

SUBROUTINES CALLED: READIN

CALLED BY: Entered through TSS subsystem YFORT

Method:

First the file THEJOB is attached and the IDENT card added to it. Next the mode is requested. If the reply is 'RUN', skip to statement 200 (figure 197). If the reply in 'COMPILE' the program asks for a list of the modules changed. A reply of 'HELP' will list the modules. A reply of 'UTILITY' or 'UTIL' causes '631IDPOO/TEST/COP/CANOF/UTIL' to be added to THEJOB. Now the program goes through all modules adding files from the CANOF catalogue for unchanged modules and from the NEWCANOF catalogue for changed modules. Then the user is asked if run mode is desired. If not skip to statement 300 (figure 197).

Statement 200

CANOF/RUN is added to THEJOB. The VRBLIM file is attached, read in and detached. The user now inputs a list of the desired verbs. If the reply is 'HELP', the legal verbs are listed. The program now calculates the run limits. The limits are displayed and the user given the opportunity to change them. Now the user is asked to input data file names. Each file name is added to THEJOB. When the reply 'DONE' is encountered, the user is asked if additional data is desired. If the reply is no, skip to statement 300. Otherwise the user is asked to input card images which are added to THEJOB. When a blank card is input the user is asked if additional files are desired and if so control returns to the input of data files.

Statement 300

The LIMITS card is added to THEJOB plus any final cards and the user is instructed as to how to execute the JCL.

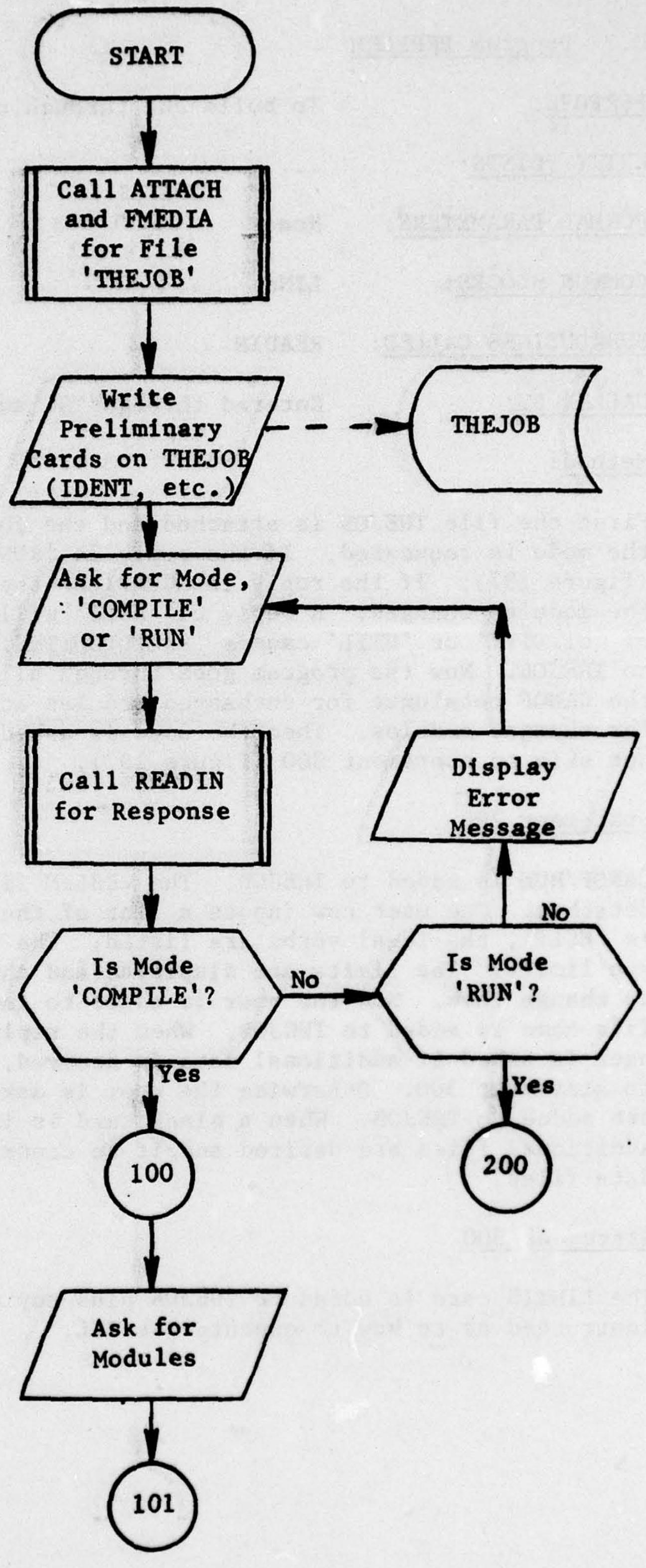


Figure 197. Program PERFORM (Part 1 of 8)

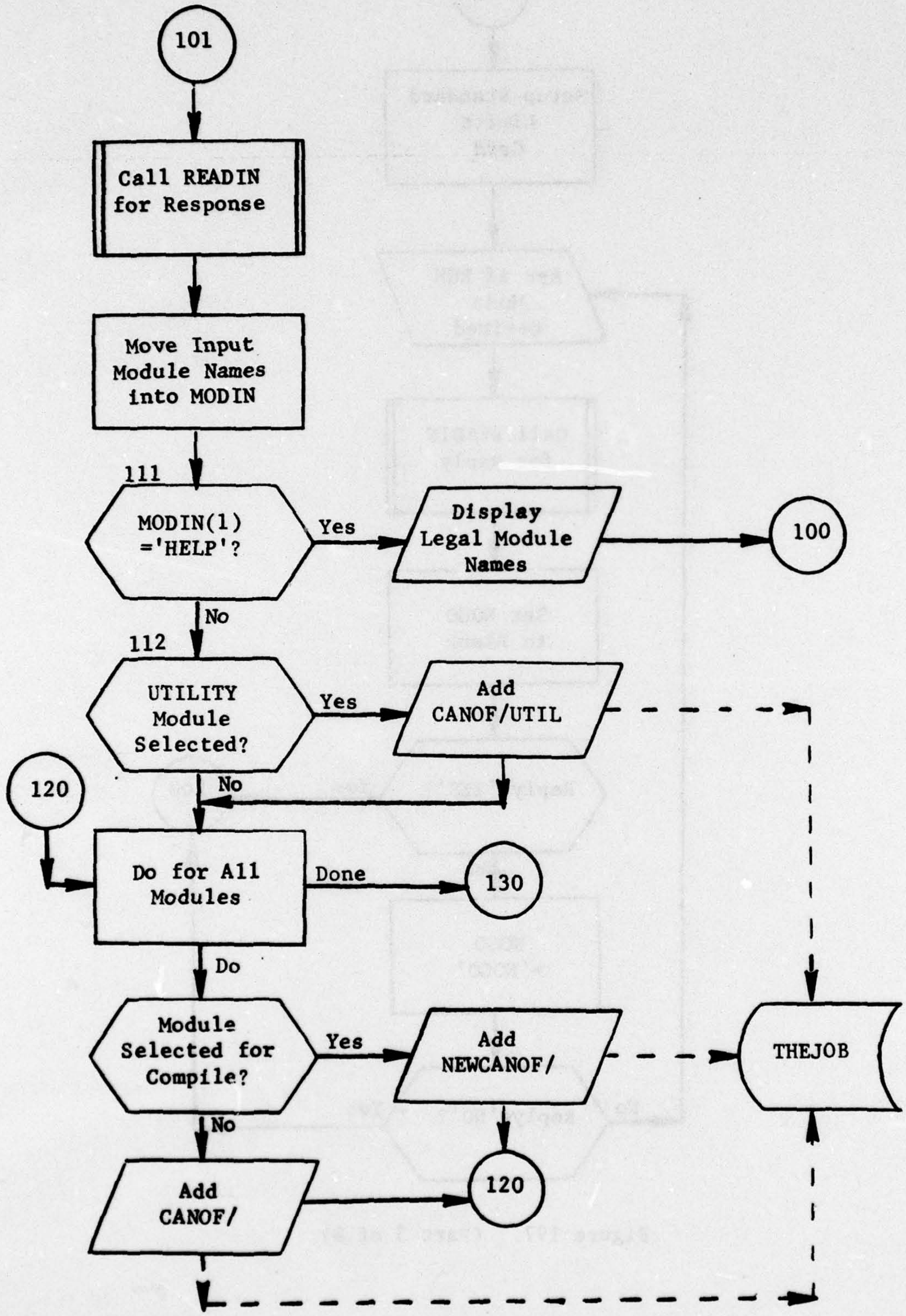


Figure 197. (Part 2 of 8)

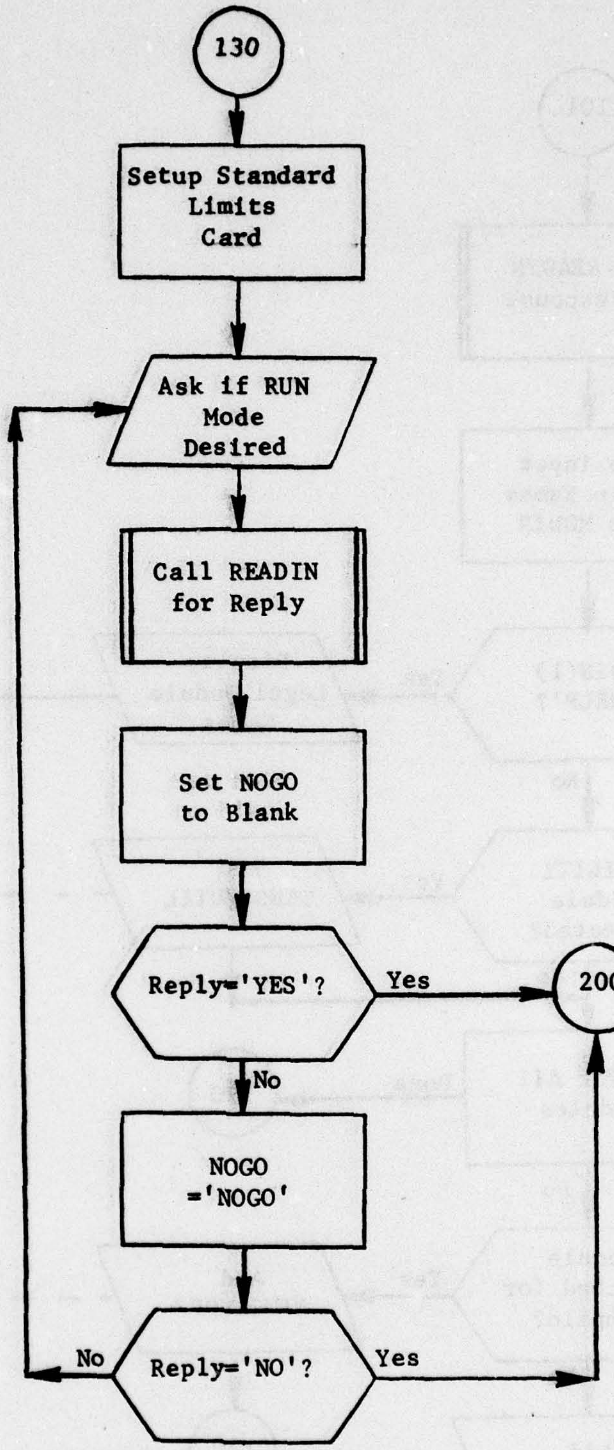


Figure 197. (Part 3 of 8)

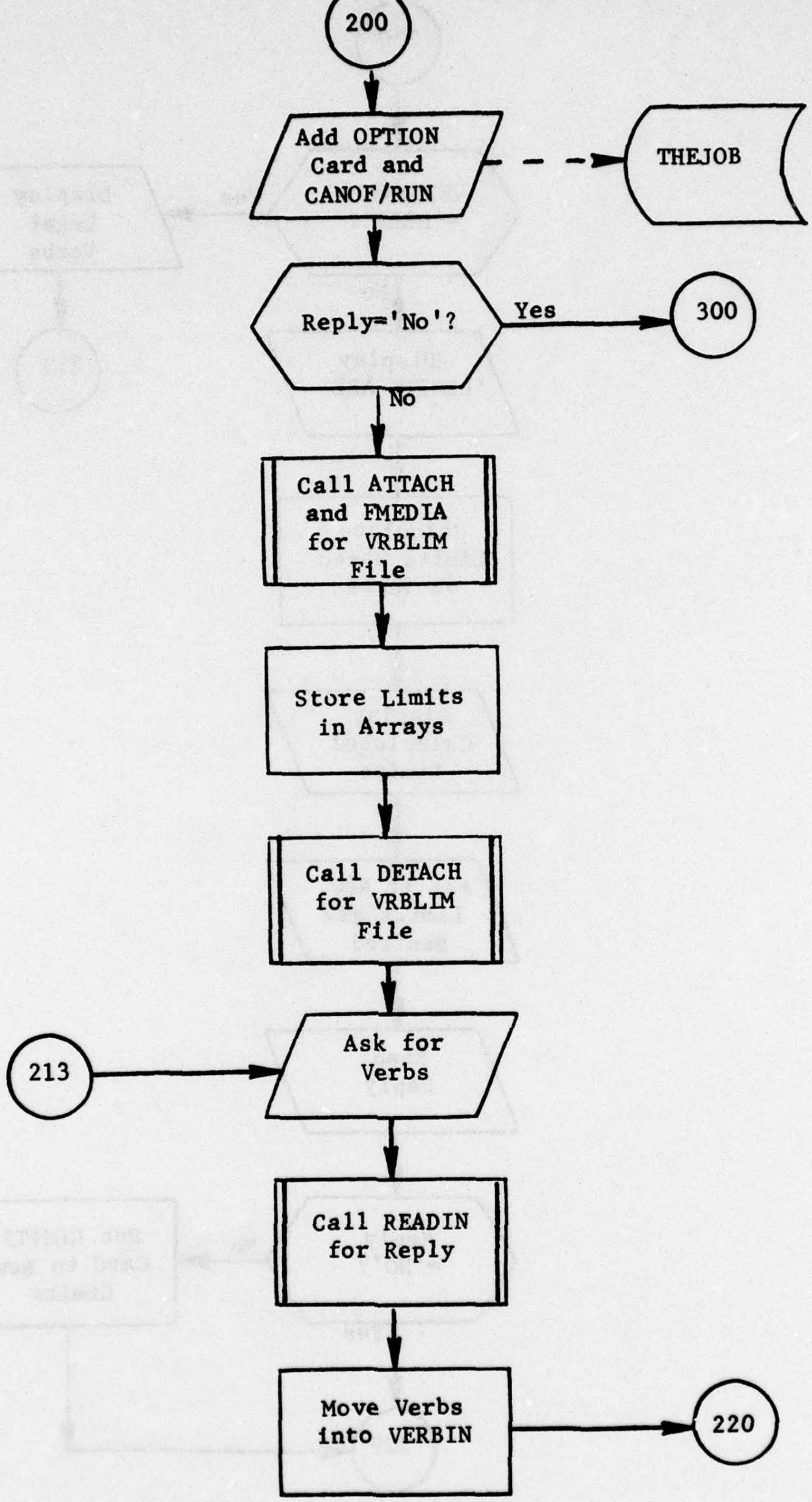


Figure 197. (Part 4 of 8)

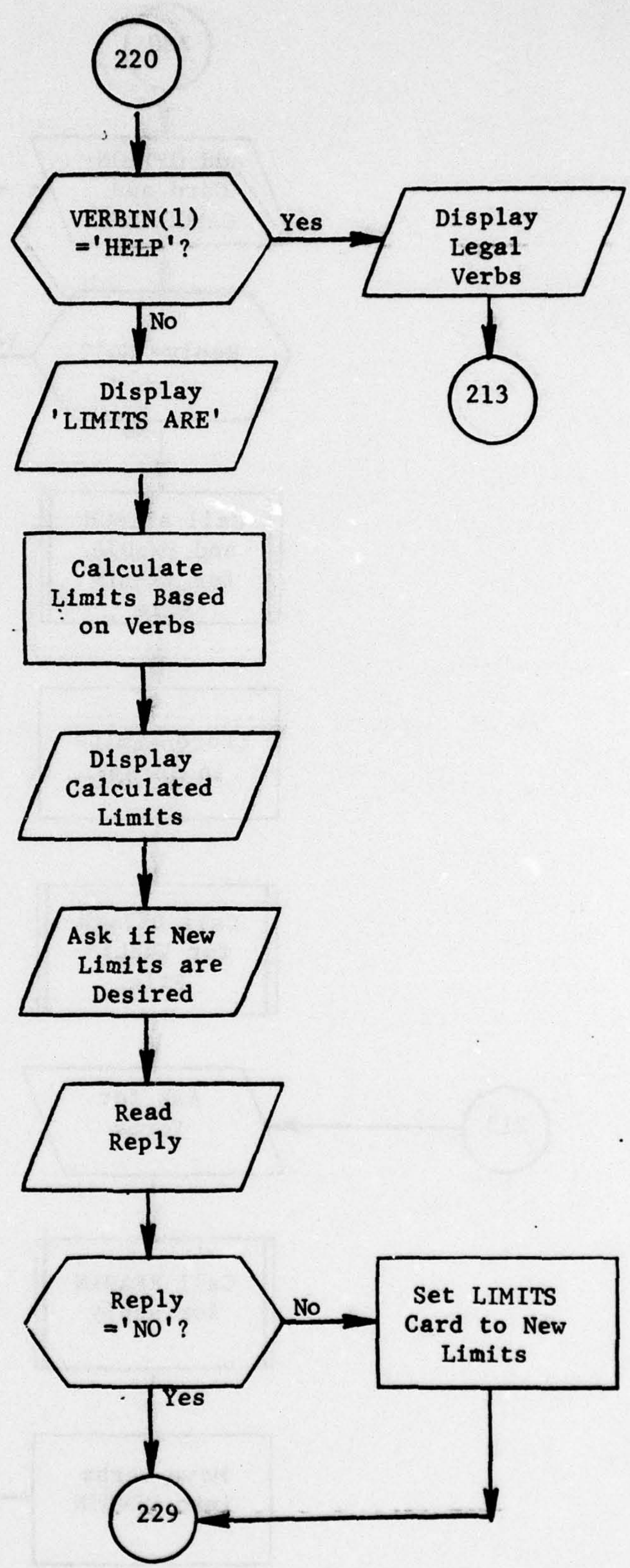


Figure 197. (Part 5 of 8)

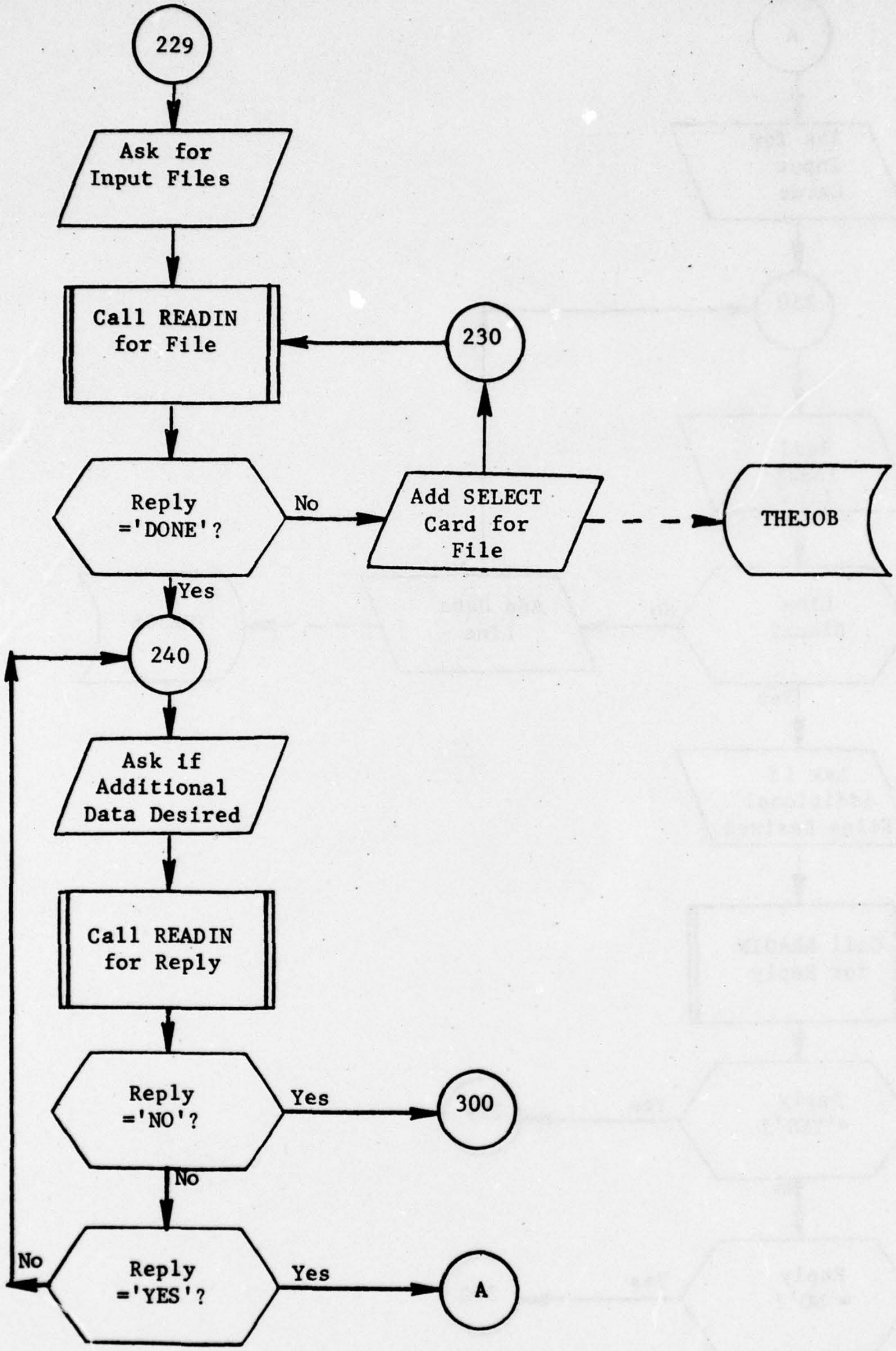


Figure 197. (Part 6 of 8)

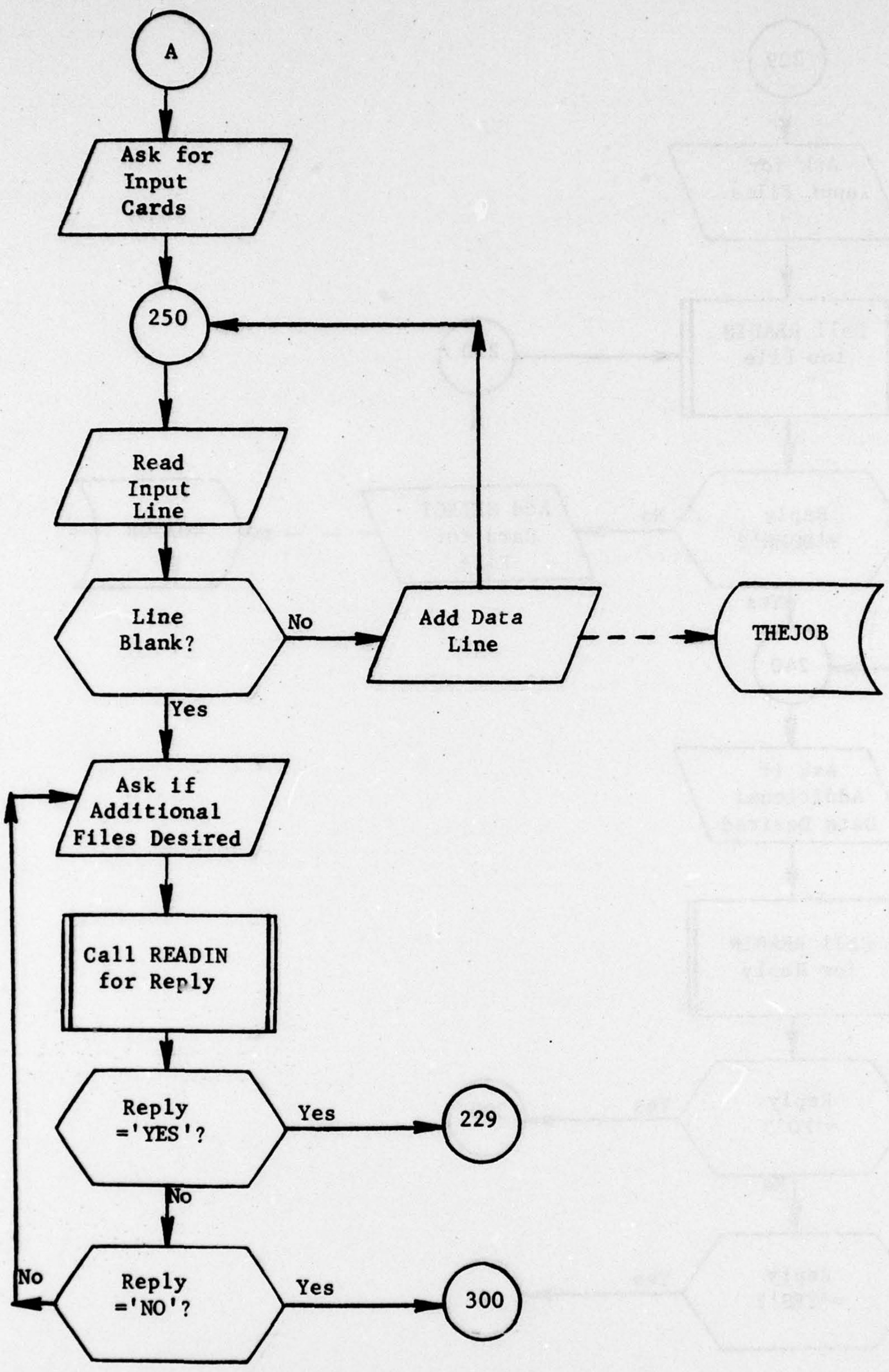


Figure 197. (Part 7 of 8)

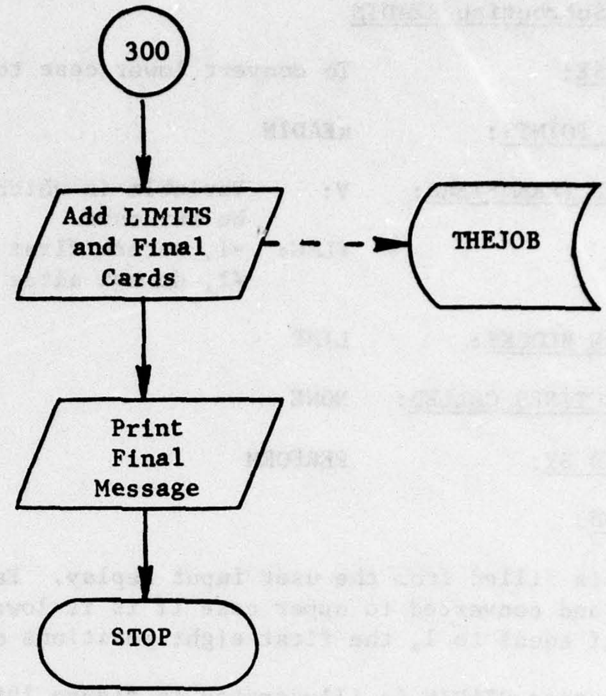


Figure 197. (Part 8 of 8)

C.8 Subroutine READIN

PURPOSE:	To convert lower case to upper case	
ENTRY POINTS:	READIN	
FORMAL PARAMETERS:	V:	Variable in which first eight characters may be returned
	FLAG:	=1, return first eight characters in V. ≠1, do not alter V.
COMMON BLOCKS:	LINE	
SUBROUTINES CALLED:	NONE	
CALLED BY:	PERFORM	

Method:

LINE is filled from the user input replay. Each character is then examined and converted to upper case if it is lower case. FLAG is checked and if equal to 1, the first eight positions of LINE are encoded into V.

Subroutine READIN is illustrated in figure 198.

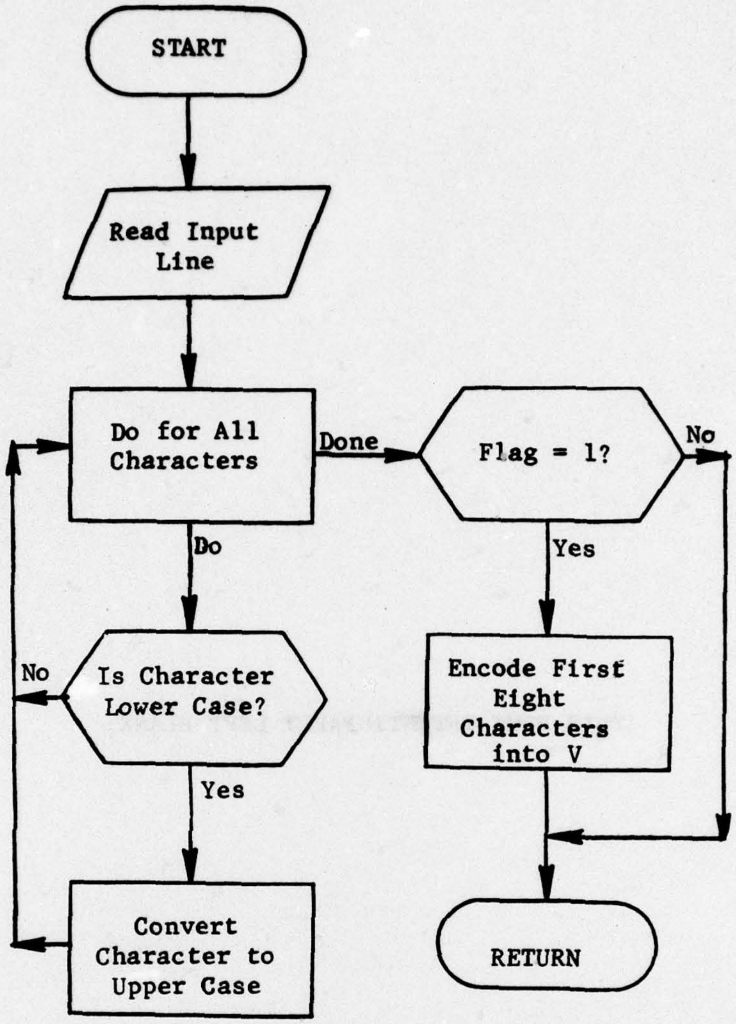


Figure 198. Subroutine READIN

Preceding Page Blank - NOT FILMED

DISTRIBUTION

Addressee	Copies
CCTC Codes	
Technical Library (C124)	3
C124 (Stock)	6
C126	1
C313	1
C314	11
C600	1
C730	1
DCA Code	
205	1
EXTERNAL	
Chief, Studies, Analysis and Gaming Agency, OJCS ATTN: SFD, Room 1D957, Pentagon, Washington, DC 20301	5
Chief of Naval Operations, ATTN: OP-96C4, Room 4A478, Pentagon, Washington, DC 20350	2
Commander-in-Chief, North American Air Defense Command ATTN: NPXYA, Ent Air Force Base, CO 80912	2
Commander, U. S. Air Force Weapons Laboratory (AFSC) ATTN: AFWL/SAB, Kirtland Air Force Base, NM 87117	1
Commander, U. S. Air Force Weapons Laboratory (AFSC) ATTN: AFWL/SUL (Technical Library), Kirtland Air Force Base, NM 87117	1
Director, Strategic Target Planning, ATTN: (JPS), Offutt Air Force Base, NE 68113	2
Defense Documentation Center, Cameron Station, Alexandria, VA 22314	12
	50

Preceding Page Blank - Not Filmed

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CSM MM 9-77 Volume I, Parts I & II	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK), Program Maintenance Manual, Data Management Subsystem		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Dale J. Sanders Paul F. M. Maykrantz Jim M. Herrin Edward F. Bersson		8. CONTRACT OR GRANT NUMBER(s) DCA 100-75-C-0019
9. PERFORMING ORGANIZATION NAME AND ADDRESS System Sciences, Incorporated 4720 Montgomery Lane Bethesda, Maryland 20014		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Command and Control Technical Center Room BE-685, The Pentagon, Washington, DC 20301		12. REPORT DATE 1 June 1977
		13. NUMBER OF PAGES 956
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for public release; distribution unlimited.

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

War Gaming, Resource Allocation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The computerized Quick-Reacting General War Gaming System (QUICK) will accept input data, automatically generate global strategic nuclear war plans, provide statistical output summaries, and produce input tapes to simulator subsystems external to QUICK.

The Program Maintenance Manual consists of four volumes which facilitate maintenance of the war gaming system. This volume, Volume I, provides the programmer/analyst with a technical description of the purpose, functions, general procedures, and programming techniques applicable to the modules and subroutines

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT (Cont'd)
of the Data Management Subsystem.

The Program Maintenance Manual complements the other QUICK Computer Manuals to facilitate application of the war gaming system. These manuals (Series 9-77 for Volumes I & II, Series 9-74 for Volumes III & IV) are published by the Command and Control Technical Center (CCTC), Defense Communications Agency (DCA), The Pentagon, Washington, DC 20301.